

QAVSA: Question Answering Using Vector Symbolic Algebras

by

Ryan Laube

A thesis
presented to the University Of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2024

© Ryan Laube 2024

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

With the advancement of large pretrained language models (PLMs), many question answering (QA) benchmarks have been developed in order to evaluate the capabilities of these models. Augmenting PLMs with external knowledge in the form of Knowledge Graphs (KGs) has been a popular method to improve their question-answering capabilities, and a common method to incorporate KGs is to use Graph Neural Networks (GNNs). As an alternative to GNNs for augmenting PLMs, we propose a novel graph reasoning module using Vector Symbolic Algebra (VSA) graph representations and a k -layer MLP. We demonstrate that our VSA-based model performs as well as QA-GNN, a model combining a PLM and a GNN-module, on 3 multiple-choice question answering (MCQA) datasets. Our model has a simpler architecture than QA-GNN, converges 37% faster during training, and has constant memory requirements as the size of the knowledge graphs increase. Furthermore, a novel method to analyze the VSA-based outputs of QAVSA is presented.

Acknowledgements

This work would not exist without the wonderful guidance of my supervisor, Chris Eliasmith. Many invaluable discussions and feedback came from my lab mates in the Computational Neuroscience Research Group (CNRG) and the other members in the Centre for Theoretical Neuroscience (CTN).

This work was supported by CFI (52479- 10006) and OIT (35768) infrastructure funding as well as the Canada Research Chairs program, and NSERC Discovery grant 261453.

Dedication

This is dedicated to my partner, Kyle, who helped to keep me sane throughout my research and stayed a grounding presence for me while in Waterloo. I would also like to thank my brother, Carter, and my parents, who have always supported and guided me to be the best version of myself.

Table of Contents

Author’s Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	viii
List of Tables	ix
1 Introduction	1
2 Background	3
2.1 Large Language Models (LLMs)	3
2.1.1 LLM Families	4
2.1.2 LLM Capabilities and Limitations	6
2.1.3 Bidirectional Encoder Representations from Transformers (BERT) Models	8
2.2 Knowledge Graph Question Answering	9
2.3 Vector Symbolic Algebras	10
2.3.1 Applications of Vector Symbolic Algebras	12

2.3.2	Holographic Reduced Representations (HRR)	13
2.3.3	Vector-Derived Transformation Bindings (VTB)	16
2.3.4	Use of VSAs with LLMs For Question Answering	18
3	Methodology	19
3.1	QAVSA Architecture	20
3.1.1	Vector Symbolic Algebra Representation of the Knowledge Graph	20
3.1.2	PLM Encoding of the QA-context	21
3.1.3	Feed Forward Layers	22
3.2	Datasets and Knowledge Graphs	23
3.3	Knowledge Graph Entity Embeddings	25
3.4	Knowledge Graph Pre-Processing	26
3.5	Model Variations	26
3.6	Training Details	29
3.7	Final Hyperparameters	33
4	Results	36
4.1	Main Results	36
4.2	QAVSA Variations and Ablations	39
4.3	Graph Embedding Analysis	44
4.3.1	Case 1: Correct prediction and relevant triples	45
4.3.2	Case 2: Correct prediction and irrelevant triples	47
4.3.3	Case 3: Incorrect prediction and relevant triples	48
4.3.4	Case 4: Incorrect prediction and irrelevant triples	49
5	Conclusion	50
	References	52

List of Figures

3.1	QAVSA model outline. The QA context, $[q a]$, is input to a PLM to generate an LM encoding for the context and is also used to generate a KG subgraph. The LM encoded QA context is added to the graph, and the graph is converted to a single vector using a VSA. The VSA representation is fed through a k-layer MLP, concatenated with the original QA context encoding, and passed through a single FF layer to score the QA pair. . . .	19
3.2	The feedforward layer architecture in QAVSA with the graph VSA vector, \mathbf{g} , and LM QA-context, \mathbf{z} , as input. The dimensionality of the weight matrices in the Linear modules are shown, and the probability of dropout is shown with the Dropout modules.	22
3.3	The three strategies for placing residual connections in QAVSA are: a) a single connection from the input graph vector, g , to last layer; b) connections between consecutive layers; c) connections from the input graph vector to each layer.	29
3.4	Examples of the different types of learning rate schedules tested. For these particular examples, the cosine and cosine with restarts schedules have 2 cycles, and the polynomial decay is quadratic (power of 2).	31
4.1	Mean accuracy of QA-GNN, QAVSA using our tuned LR schedule, and QAVSA using the LR schedule of QA-GNN on the CSQA dev. split for each epoch, averaged over 5 seed runs.	37

List of Tables

3.1	Example questions from the development split of CSQA, OBQA, and MedQA.	24
3.2	Example facts from the OBQA fact sheet.	25
3.3	Summary of the QAVSA model variations that were tested.	27
3.4	Hyperparameter tuning on CSQA using Tree-structured Parzen Estimator as a sampler, median pruning rule with 20 startup trials and 6 warmup steps per trials. 195 trials were completed. QAVSA used L2-normalized embeddings, HRR VSA graphs with TZW concept embeddings, random relation embeddings and a linear decay LR schedule.	30
3.5	Hyperparameter tuning on CSQA using Tree-structured Parzen Estimator as a sampler, median pruning rule with 20 startup trials and 15 warmup steps per trials. 90 trials were completed. HRR VSA graphs with TZW concept embeddings and random relation embeddings were used.	32
3.6	Hyperparameter tuning on CSQA using Tree-structured Parzen Estimator with 10 startup trials as a sampler and no pruning. 135 trials were completed. HRR CSA graphs with no QA-context node added, TZW concept embeddings, and BERT relation embeddings were used.	33
3.7	Hyperparameter tuning on OBQA using Tree-structured Parzen Estimator with 10 startup trials as a sampler and no pruning with 150 trials completed. HRR VSA graphs with BERT concept and relation embeddings were used, and the LR schedule was Cosine w/ restart.	34
3.8	Hyperparameter tuning on MedQA with PLM weights frozen.	34
3.9	Hyperparameter tuning on MedQA with unfrozen PLM weights.	34
3.10	The best model parameter values on all experiment datasets discovered by hyperparameter optimization.	35

4.1	Accuracy and 95% confidence interval on CSQA inhouse dev. and test splits. Reproduced results (*) use reproduced node embeddings and all results are averaged over 5 different seeds.	36
4.2	Accuracy and 95% confidence interval on OBQA dev. and test splits. Reproduced results (*) use reproduced node embeddings and all results are averaged over 5 different seeds.	38
4.3	Test accuracy and standard deviation on MedQA. Reproduced results are denoted with * and all results are averaged over 3 runs.	38
4.4	Dev. and test accuracy on CSQA with different VSA MLP dimensions. The input dimension for all trials is 1024.	39
4.5	Accuracy on CSQA and OBQA with different combinations of PLM QA-context encodings and graph concept/relation embeddings.	40
4.6	Accuracy (%) of QAVSA on the dev. and test splits of MedQA with varied QA-context and knowledge graph entity embeddings.	41
4.7	Accuracy of different model variations on dev. and test splits of CSQA. . .	41
4.8	Accuracy of different model variations on dev. and test splits of OBQA. . .	42
4.9	Accuracy of different model variations on dev. and test splits of MedQA. . .	43

Chapter 1

Introduction

Models that perform question answering require some amount of knowledge, whether it is structured or implicit, about the concepts to be reasoned over. With large language models (LLMs), linguistic knowledge is implicit in the token embeddings that have been learned using a self-attention mechanism on large text corpora to perform next-token prediction [82]. With enough model parameters and training data, these LLMs have been successful in a wide range of question-answering and reasoning benchmarks. However, when analyzing reasoning performance, i.e. the ability to learn and generalize from logic rules, LLMs, especially smaller ones like BERT and RoBERTa, demonstrate inconsistent performance [97].

As a result, there have been many studies that work to integrate external, structured knowledge and reasoning modules with pretrained language models (PLMs) to perform more reliable logical reasoning and question answering. One type of knowledge structure that is commonly used are knowledge graphs (KG), due to their suitability for symbolic reasoning [49]. Graph Neural Networks (GNNs) are deep learning networks that have gained popularity for reasoning over graph-structured data. Consequently, methods that integrate KGs with PLMs often also use GNNs [96]. One recent approach to combining these techniques is captured by the QA-GNN model [95], which is able to answer multiple-choice questions by scoring the plausibility of each question answer.

Rather than using GNNs to model structured data, our model, QAVSA, uses a different method, Vector Symbolic Algebras (VSAs; also known as Vector Symbolic Architectures), to represent and combine high-dimensional concept vectors in a structured way. The integration of PLMs and VSAs has not been previously proposed, to our knowledge. As a result, the focus of this thesis is to perform a comparison between our VSA-based method and

the GNN-based method of QA-GNN for improving reasoning capabilities of PLMs in the context of multiple-choice question answering. Specifically, we compare the performance of QAVSA and QA-GNN on CommonsenseQA [78], OpenbookQA [60], and MedQA-USMLE [39], with the first two datasets being focused on commonsense reasoning and the latter focusing on domain-specific medical questions.

The main contributions of this thesis are:

1. A novel combination of PLM text encoders and VSAs that performs as well as an analogous GNN-based method on three MCQA datasets while training 37% faster.
2. A comparative evaluation of different VSAs and PLMs used for QAVSA.
3. A comparison on several variations of the QAVSA architecture.
4. A new VSA-specific method of analyzing model explainability, which is applied to the output graph embeddings of QAVSA.

The thesis outline is as follows:

- Chapter 2 provides a literature review of the current state of LLMs, question answering, and VSAs.
- Chapter 3 outlines the QAVSA model architecture, experiments performed, and training details of the model.
- Chapter 4 provides the results and discussion of the experiments and model variations outlined in Chapter 3. Further analysis is done on the VSA representations with several case studies of QAVSA outputs discussed.

As a note, portions of Chapters 1, 2, 3, 4, and 5 contain content from a previously published paper at the workshop on Representation Learning for NLP (RepL4NLP) [50].

The code repository for this thesis is available at <https://github.com/r1aube/qaspa>.

Chapter 2

Background

This chapter contains of an overview of the current state of Large Language Models architectures, families, and their applications in Sections 2.1.1 and 2.1.2, with particular emphasis on BERT models in Section 2.1.3. This is followed by some background on Knowledge Graph Question Answering (KGQA) in Section 2.2, which is the main task QAVSA is applied to (see Chapter 3). Finally, we present an overview of Vector Symbolic Algebras (VSAs) (Section 2.3), starting with the applications of VSAs (Section 2.3.1), followed by a review of Holographic Reduced Representations (HRRs) and Vector-Derived Transformation Binding (VTB) in Sections 2.3.2 and 2.3.3. Portions of Sections 2.1.3, 2.2, and 2.3 include content from a previously published paper at the RepL4NLP workshop [50].

2.1 Large Language Models (LLMs)

Since the invention of the Transformer [82], the field of Natural Language Processing (NLP) has shifted dramatically. Transformers allowed for highly parallelized training of language models, which was not possible before with recurrent networks like LSTMs, and inspired models with impressive performance scaling on NLP and downstream tasks with an increase in model size and training data [100]. This has led many companies to develop Large Language Models (LLMs) that have up to trillions of parameters trained on over 10 trillion tokens that have state of the art performance on tasks like question answering, natural language understanding, and code generation, along with many others [100].

The main component of the Transformer architecture is the self-attention block that learns associations between sequential text tokens through query, key, and value vectors

for each token in the sequence. Using initial input token embeddings along with positional encodings, query, key, and value matrices are learned through training. These matrices generate attention-aware output embeddings for each token. Combining multi-head attention blocks, which are blocks that perform this attention operation several times concurrently, with feed forward Neural Network layers, make up encoder and decoder blocks that can be stacked. Full details of the Transformer architecture can be found in [82]. For models with only the decoder component, like the GPT model, an auto-regressive text generator, a linear projection and Softmax of the output embeddings is computed to determine the next most probable token in the sequence [61]. For models with only the encoder component, like the BERT model, the final hidden layer embeddings are used for downstream tasks like classification.

The impressive generalization capabilities exhibited by large Transformer-based language models, such as in-context learning and instruction following, have led them to be treated as foundation models that can be applied to, or finetuned for, many downstream tasks. Models like GPT-3 display in-context learning, which is when a model can generate reasonable output for some given input task with up to a few examples of the task and without any further training [61]. Instruction tuning is a widespread technique, where different tasks are formatted as text instructions, to increase LLM performance in instruction following for unseen tasks [61]. LLM performance on tasks involving reasoning has also been improved with prompting strategies like Chain of Thought (CoT) that instruct models to output reasoning steps instead of just the final answer [100].

2.1.1 LLM Families

There are generally three types of Transformer-based architectures: encoder-only, decoder-only, and encoder-decoder models. Notable encoder-only models include Bidirectional Encoder Representations from Transformers (BERT) and its successors RoBERTa, DeBERTa, etc., which are smaller language models with up to 340 million parameters, developed by Google [14, 57]. Since there is no decoder portion of the model, they are not suited for text generation, but instead are suited to sentence classification and question answering [61]. This is the family used in our work. Decoder-only, or auto-regressive, models only use previous text tokens in a sequence when generating new tokens, and are often pre-trained using next-token prediction. The most popular decoder-only model is the Generative Pre-trained Transformer (GPT) family of models. Encoder-decoder models are useful for sequence-to-sequence tasks like summarization or translation, and some notable models of this variety are Text-to-Text Transfer Transformer (T5), which uses transfer learning that

casts all NLP tasks as sequence-to-sequence tasks [61, 66], and BART, which combines Bidirectional and Auto-Regressive Transformers [51].

Some popular examples of LLMs are the GPT series of models developed by OpenAI, the LLaMA series of models developed by Meta, and the PaLM series of models developed by Google. The Generative Pre-trained Transformer (GPT) series includes models such as GPT-3, ChatGPT, GPT-4, and Codex. GPT-3, a 175 billion parameter model pre-trained on 500 billion tokens [100, 61], demonstrated few shot learning on a wide variety of NLP tasks. Using Reinforcement Learning from Human Feedback (RLHF) to tune GPT-3 enabled the model to better follow human instructions in a truthful and safe manner, and led to the popular chat bot ChatGPT. Codex is a GPT model trained on billions of lines of code in order to generate code. The newest and most powerful GPT model, GPT-4, is also multi-modal, with the ability to take text and image as inputs [61].

The LLaMA series from Meta includes decoder-only models such as LLaMA-1, LLaMa-2, LLaMA-2 Chat, and Mistral. The initial LLaMA models have parameter counts ranging between 7B and 65B and were pre-trained on more tokens than typically used at several model sizes [61]. LLaMA-2 Chat was trained using RLHF, like ChatGPT, after its pre-training and supervised fine-tuning stages [80]. Mistral is a 7B parameter model that processes text sequences with arbitrary length at reduced inference costs using sliding window attention, and outperforms the 34B parameter version of LLama-2 in most benchmarks, including reasoning and mathematics tasks [38].

The PaLM family from Google includes decoder-only models like U-Palm, Med-PALM, and PaLM-2. PaLM is a 540B parameter model pre-trained on 780 billion tokens that produced state-of-the-art results on many benchmarks, including the BIG-bench benchmark, which focuses on capturing reasoning abilities [61]. U-PaLM is a model that has computational savings of approximately 2x compared to PaLM by using UL2R, a training method that extends the regular pre-training by a small amount of steps with a mixture-of-denoiser objective [79]. PaLM-2 outperforms PaLM in its reasoning and multilingual performance while being more computationally efficient, with only 16B parameters pre-trained on 100B tokens [100]. Med-PALM is finetuned using instruction prompt tuning in order to provide reliable outputs in the medical domain, and Med-PALM 2, a further iteration on Med-PaLM that is trained with ensemble prompting, produced state-of-the-art results on MedQA, a question-answer dataset based on professional medical tests [71].

2.1.2 LLM Capabilities and Limitations

Traditional NLP tasks are still common methods used to evaluate LLMs, and these include, but are not limited to, sentiment analysis, text classification, natural language inference, semantic understanding, and translation [8]. Some NLP datasets that are commonly used to evaluate LLMs are General Language Understanding Evaluation (GLUE) [86], which includes tasks like similarity, paraphrasing, and natural language inference (NLI), its more difficult variant SuperGLUE [85], and WinoGrande, a dataset consisting of pronoun resolution problems [69].

Benchmarks that aggregate various downstream tasks have been useful in evaluating the capabilities and limitations of LLMs. Holistic Evaluation of Language Models (HELM) is a benchmark intended to improve transparency of LLMs by measuring their robustness, fairness, toxicity, bias, etc., across 16 broad scenarios [52]. Massive Multitask Language Understanding (MMLU) is designed to test LLM knowledge in STEM, humanities, problem solving, and many others across 57 tasks in zero-shot and few-shot scenarios (hendrycks). Huge models like Gemini Ultra-1760B [1] have the best performance on MMLU, reaching an accuracy of 90.4%, where expert human performance is determined to be 89.8%. BIG-Bench is an aggregate of 204 tasks including traditional NLP tasks, logic and math, pro-social behaviour, and scientific understanding tasks [75]. A subset of 23 tasks from BIG-bench that LLMs under-performed on compared to the average human tester, labeled BIG-bench hard, is often used to evaluate newer models, and it was determined that Chain of Thought (CoT) prompting enabled Codex to outperform the average human tester on 17 of the 23 tasks [77].

Many application-specific datasets have also been developed to test specific functionality and knowledge of LLMs. Coding is one popular application of LLMs, with evaluation datasets such as Mostly Basic Python Programming (MBPP) [3] and HumanEval [9], which measures functional correctness and knowledge of programming fundamentals using Python code generated by LLMs. Healthcare is another well-discussed use-case of LLMs, such as ChatGPT being used for medical queries and as an assistant, although there are limitations regarding misdiagnosis, uncertainty, and privacy [8]. New LLMs are also often tested on professional exams, like the United States Medical licensing Examination (USMLE) in the medical field and a simulated version of the bar exam in the field of law [8].

Many reasoning and knowledge testing datasets come in the form of multiple-choice question-answering (MCQA). Some examples of MCQA datasets are: CommonsenseQA (CSQA), which tests reasoning about everyday objects and scenarios [78]; HellaSwag, which tests commonsense NLI [98]; OpenbookQA (OBQA), which evaluates highschool level sci-

ence knowledge [60]; AI2’s Reasoning Challenge (ARC), which tests on elementary science questions involving reasoning [11]; and Social Interaction QA (SIQA), which benchmarks social intelligence [70]. High performance can come from fine-tuning on MCQA datasets as has been shown with UnifiedQA [43] and UNICORN [58]. Superior performance with these models has also been shown to be achievable with prompting techniques including few-shot prompting [1], Chain-of-Thought prompting with self-consistency [35], and ensemble refinement [71].

Although LLMs are capable in many downstream tasks, LLMs can be unreliable due to their hallucinatory behaviour and inconsistency in reasoning. It has been observed that even the most capable LLMs like GPT-4 hallucinate facts, and demonstrate an over-confidence in their outputs [100]. Logical inconsistency can come from invalid reasoning processes, incorrectly referencing previous reasoning steps to produce a wrong answer, or producing a correct answer that is irrelevant to its reasoning output [100]. Apart from factual reliability, LLMs have demonstrated social biases and toxicity in their output because of the inclusion of these biases in their training data [100]. TruthfulQA [54] is one useful benchmark on reliability that tests a model’s ability to distinguish fact from false beliefs and misconceptions that humans may have.

It is difficult to inject LLMs with specialized knowledge, as it can cause them to struggle in certain areas, create catastrophic forgetting, and reduce their in-context learning ability [100]. Similarly, it is difficult and expensive to finetune LLMs to update their knowledge consistently for tasks that require knowledge that is outside of an LLM’s training distribution. This challenge, along with LLMs struggle to represent structured data for tasks like knowledge-base question answering (KBQA) [100], has led researchers to integrate and augment LLMs with external knowledge sources, especially structured knowledge (like knowledge graphs), through methods like [100] (RAG) and Graph Neural Networks [26].

LLMs also particularly struggle with abstract reasoning. Gendron et al. [29] find that several LLMs, including GPT-4 and LLaMA2-7B have very limited performance on several datasets including visual question answering, solving Raven’s progressive matrices, and a subset of BIG-Bench tasks, even with heavy reliance on prompting and refinement techniques. Wu et al. [90] find that GPT-4 and PaLM-2 performance on 11 tasks that include spatial and syntactic reasoning, arithmetic, programming, etc., consistently degrades on counterfactual variants of these tasks. Current LLM deficiencies in reasoning have led others to consider integrating LLMs with symbolic and mathematical reasoning models. A successful example of this type of model is AlphaGeometry, which uses an LLM to guide a symbolic deduction engine and is able to solve Olympiad-level geometry problems [81].

A limitation within the academic field of studying and developing LLMs is the sheer

amount of data and compute resources required to train these models, especially when many state-of-the-art LLMs are hundreds of billions of parameters, which only large companies have the resources to develop. As a result, an effort is being made to develop smaller models that are more efficient in terms of architecture and training methods, along with models that use external tools or knowledge for specific downstream tasks [100].

2.1.3 Bidirectional Encoder Representations from Transformers (BERT) Models

The pre-trained language models (PLMs) used to generate embeddings for our QAVSA model are based on Bidirectional Encoder Representations from Transformers, BERT [14]. As noted above, BERT consists in a tokenizer that converts text input into a sequence of tokens, each with their own embedding vector from WordPiece, plus a stack of Transformer layers. BERT is an encoder-only model that focuses on modelling input sequences, as opposed to models with a decoder component like GPT that are trained to also generate output tokens. This has led encoder-only models like BERT to be a popular choice for downstream tasks like MCQA, sentiment analysis, and sentence classification, rather than text generation [61].

The BERT model we use is pretrained to learn bidirectional representations of text using masked language modelling (MLM), also known as a Cloze task, and next sentence prediction as training objectives [14]. The MLM task is to predict the vocabulary ID of randomly masked input tokens. Language models that are trained with next-token prediction like GPT are unidirectional, since tokens are generated in sequence. MLM avoids this since the token classification is conditioned on the both the left and right contexts. Next sentence prediction (NSP) pretraining consists of predicting the entailment of two sentences, A and B, where B is the actual sentence that follows A for 50% of the samples and is another random sentence from the corpus for the other 50%. Each input token sequence into BERT begins with a [CLS] token and consists of a single sentence, or a pair of sentences with a [SEP] token between them. The final token representations from the final Transformer layer can be input to another layer for token-level tasks, and the final [CLS] representation is treated as the aggregated sequence representation for sentence-level classification tasks.

Next sentence prediction improves BERT’s performance on QA and NLI datasets such as Stanford Question Answering Dataset v1.1 (SQuAD), Question-Answering Natural Language Inference (QNLI), and Multi-Genre Natural Language Inference (MNLI) [14]. SQuAD is a QA dataset that consists of paragraph-question pairs where the Wikipedia-

derived paragraph contains the answer to the question [67]. QNLI is derived from SQuAD v1.1 that converts the form of the task to sentence pair classification by asking whether the context sentence provides the answer to the question [86]. This conversion of format removes the assumption that the context always contains the answer, but also removes the requirement that the model must provide the actual answer. MNLI is a crowd-sourced set of hypothesis-conclusion sentence pairs with the goal of determining whether the conclusion entails, contradicts, or is neutral to the hypothesis [89].

Another configuration of BERT, named Robustly optimized BERT approach (RoBERTa), has the same architecture as BERT, but uses a Byte-Pair Encoding (BPE) tokenizer relying on subword units, and follows a different pretraining scheme. RoBERTa [57] is trained on larger more diverse data than BERT, pre-trains on longer text sequences, uses dynamic masking rather than static token masking so that the same text samples are masked in different places in the sequence, is not pretrained on the NSP task. This more in-depth training procedure has generally led RoBERTa to be more proficient in downstream tasks such as paraphrase detection and sentiment classification.

BERT, RoBERTa, and BioLinkBERT, a BERT model trained in the biomedical domain with citation links [94], are used to generate embeddings at the word and sentence-level in several components of QAVSA (see Sections 3.1.2 and 3.3).

2.2 Knowledge Graph Question Answering

Knowledge graphs are a type of structured knowledge that are defined in terms of vertices, or nodes, that represent objects and concepts and edges that connect or relate concepts together. The edges can be bidirectional if it just represents a numerical weighting or connection between nodes, but they can also be directed if the edges are relational, representing relations such as “belongs to” or “has a”. There has been relevant Machine Learning methods that process graphs, such as Graph Neural Networks, which have been applied to traffic forecasting, language processing, and drug discovery [96, 101].

Several approaches have been proposed to integrate external knowledge graphs with PLMs in order to make use of more domain-specific structured knowledge. These include KEAR [91] and DEKCOR [92], which use a PLM and knowledge retrieved from an external KG and Wiktionary to train an attention mechanism on these external knowledge bases and PLM representations to improve commonsense reasoning.

Similarly, KagNet [53] performs commonsense reasoning by grounding the concepts within each question-answer (QA) pair of multiple-choice datasets to extract subgraphs

from an external knowledge graph. KagNet then uses a combination of Graph Convolutional Networks, LSTM-based relational path encodings, and a path-based attention mechanism to identify important reasoning paths to generate graph vector encodings for each QA pair to subsequently score them. Adopting similar graph preprocessing to KagNet, MHGRN [21] performs multi-hop, multi-relation reasoning by using multi-hop message passing from Relational Graph Convolutional Networks, structured relational attention, and node attention pooling to generate its graph representations and score QA pairs.

Another family of models that also use the same graph processing above stem from the QA-GNN [95] model. QA-GNN fuses a PLM representation of the QA context as a node into the QA subgraphs. Using a Graph Attention Network (GAT), QA-GNN updates the subgraph concept embeddings, including the QA context node and edge weights. The initial PLM QA context representation, along with the final graph representation of the QA context and graph concept attention pooling is used to score the QA pairs. This method is refined in GreaseLM [99], in which the PLM token and graph node modalities of the QA context are mixed over several layers to simultaneously update the PLM and GNN concept embeddings. Further refinements are made in DRAGON [93], where the cross-modal encoder from GreaseLM is pretrained in a self-supervised fashion to perform both masked language modelling and KG link prediction. Our QAVSA model uses similar pre-processing to QA-GNN but with a different representation of the graph. Consequently, most of our direct comparisons are to QA-GNN.

There exist other models that use external KGs but are not GNN-based, such as GSC [87] and MVP-Tuning [36]. GSC uses a simple graph neural counter to reduce the node and embeddings to one dimension and performs GNN-like embedding updates on these single values. MVP-Tuning makes use of semantically similar QA pairs in the training set to improve knowledge retrieval and tunes prompt tokens of the PLM by using the QA context and retrieved KG triplets as input to the PLM.

2.3 Vector Symbolic Algebras

Vector Symbolic Algebras (VSAs) are defined by a set of four vector operations that are useful for building up and computing over structured, high-dimensional vector representations. These include the similarity, bundling (or collecting), binding, and permutation operations. The role of the similarity operator is self-explanatory: to determine how close two VSA vectors are in the vector space. A VSA generally has a set of atomic vectors that can represent any set of semantic content that one would like to compute over. Bundling these atomic vectors creates new vectors that represent a collection, or subset, of these

atomic vectors. Since the result of the bundling operator represents a simple collection of the operands, a pair of input vectors should be similar to the result of their bundling, using the defined similarity operator. On the other hand, the binding operator binds vectors to create an association between them, resulting in a vector that is dissimilar to the input vectors and any other vector in the VSA vocabulary. The role of a permutation operator is to preserve vectors during the other operations.

The reason why high-dimensional, or hyperdimensional, vectors are used in these algebras is because of the “curse” or “blessing” of dimensionality. In high-dimensional vector spaces, two vectors that are randomly chosen are quasi-orthogonal to each other, i.e., the angle between the vectors is close to 90° , and this quasi-orthogonality converges to exact orthogonality as dimensionality increases [30]. To effectively process each concept represented as a vector, or combination of vectors using VSA operations, it is critical to distinguish each vector from every other vector. Therefore, the dis-similarity, or quasi-orthogonality, that vectors have in high-dimensional spaces is crucial for representing and operating over a large number of concepts and structures, such as those found in natural language.

There are a wide variety of possible choices for these operations, but we consider only two sets of operators for QAVSA, those for Holographic Reduced Representations (HRRs; see Section 2.3.2) [64] and for Vector-derived Transformation Bindings (VTB; see Section 2.3.3) [32]. Other notable VSAs include Multiply-Add-Permute (MAP) [27] and Binary Spatter Codes (BSC) [41]. Many of these VSAs are conceptually inspired by Smolensky’s Tensor Product Representations (TPR), who implemented binding with tensor products to represent nested symbolic structures using connectionist methods [72]. The main drawback of TPR is that the binding operator does not preserve the dimensionality of the input vectors, leading to an exponential increase in vector dimension with repeated bindings. Subsequent VSA approaches avoid this problem by ensuring the operators within the VSA preserve vector dimensionality, thus allowing VSAs to represent more complex symbolic representations without the need of increasing memory or computation requirements.

Plate [64] initially proposed the HRR VSA in order to represent complex compositional structures, specifically ones used for language processing and reasoning, with distributed representations like those found in neural networks. However, there has been significant debate about whether and how neural networks can capture such structures. For instance, Jackendoff proposed four linguistic challenges involving how to neurally represent the compositional structure and rules of language [37]. These problems are:

1. *The binding problem*: Jackendoff observed that “linguistic representations must be composed from component representations”, and that many atomic representations

must be bound together to create the complex syntactic structures in human language [28].

2. *The problem of two*: How are multiple instances of the same token distinguished between each other? Consider the example sentence “The little star’s beside a big star”. Using a connectionist, i.e. neural network, representation of language where each word is represented with a specific group of neurons, a problem arises because the representation for “star” would be indistinguishable between the two occurrences of the word.
3. *The problem of variables*: We have grammatical templates where variables are constrained to classes of words such as nouns, verbs, etc. With a finite grammar that has a multitude of options for each placeholder, a grammatical template can lead to infinite possibilities for valid sentences, which is also termed productivity.
4. *Binding in working memory vs. long-term memory*: How are long-term and working memory incorporated into a cognitive model, given Jackendoff’s argument that the structures instantiated in linguistic processes are required to also be instantiated in both of these memories.

Gayler [28] argues that VSAs solve these four problems, which supports the idea that VSAs are useful in studying linguistics and reasoning within the context of neural networks. Our work here applies these ideas in the context of deep neural networks for linguistic processing by addressing the challenge of question answering. Before discussing the specifics of our proposed application, we provide applications of VSAs, followed by two specific examples of VSAs that we subsequently use in our models.

2.3.1 Applications of Vector Symbolic Algebras

The most common field to apply VSAs is in cognitive modelling. The world’s largest functional brain model, Semantic Pointer Architecture Unified Network (SPAUN), is a notable example that models various brain functions like short-term memory, motor control, visual processing, and decision making, implemented with a large spiking neural network [20, 10]. For example, SPAUN’s vision system can process hand-written digits, then store that information in a memory, and finally rewrite the digit in its by controlling an arm. HRR VSA was used to implement the memory functionality in SPAUN. Other aspects of cognition have been modeled too, such as HRR VSA-based neural networks that model emotions and how they can be influenced by bodily states and cognitive evaluations [40]. Work has

also been done relating to reasoning, such as models that perform similarly to humans in the Raven’s Progressive Matrices task, which tests abstract reasoning on geometric shapes and patterns in a 3x3 grid. Rasmussen et al. [68] developed a spiking neural network that infers rules based on learned VSA representation transformations.

There has also been an overlap between Machine and Deep Learning models and VSA methods, which are also sometimes referred to as Hyperdimensional Computing (HDC) [45]. Nunes et al. [62] developed a graph classification model by learning vertex vector embeddings for all the vertices in the graph and represent the graph by the sum of bound edges. They showed their VSA-based graph encoding method had equivalently accuracy to a GNN-based method with reduced training and inference time. Other work has used VSA-based image encoding for object classification and character recognition, visual question answering and scene understanding [44].

An extension of discrete VSAs to continuous domains was proposed with Spatial Semantic Pointers (SSPs) [48], which have been applied to spatial cognition [15] and navigation [46]. Other specific applications of SSPs and VSAs include cognitively plausible methods for representing spatial relations [59], performing biologically realistic cleanup memory in spiking neurons [76], modelling concept representation in cognition [6], spiking neural implementations of models for simultaneous localization and mapping (SLAM) [47, 16], performing operations on probability distributions [22], Bayesian optimization [25], biologically plausible Markov Chain Monte Carlo Sampling [24], modelling grid cells [15], online reinforcement learning with grid and place cell models for discrete and continuous state tasks [4, 5], and semantic mapping [17], among others.

2.3.2 Holographic Reduced Representations (HRR)

The bundling operation in HRRs is simply element-wise addition. Thus, for two arbitrary vectors, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ is defined by $S(\mathbf{x}, \mathbf{y}) = \mathbf{x} + \mathbf{y}$.

In HRRs, circular convolution is used as a binding operator, defined by

$$(\mathbf{x} \circledast \mathbf{y})_i := \sum_{j=1}^d x_j y_{((i-j) \bmod d) + 1}, i \in \{1, 2, \dots, d\}.$$

If s $\mathbf{x}, \mathbf{y} \in \mathbb{R}^4$, where $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$, then

$$\mathbf{x} \circledast \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \circledast \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} x_1y_1 + x_2y_4 + x_3y_3 + x_4y_2 \\ x_1y_2 + x_2y_1 + x_3y_4 + x_4y_3 \\ x_1y_3 + x_2y_2 + x_3y_1 + x_4y_4 \\ x_1y_4 + x_2y_3 + x_3y_2 + x_4y_1 \end{bmatrix}$$

The unbinding operation, i.e. the inverse of circular convolution, is computed using the pseudo-inverse of the given operand: $(\mathbf{x} \circledast \mathbf{y}) \circledast^{-1} \mathbf{y} \approx \mathbf{x} \circledast \mathbf{y} \circledast \sim \mathbf{y} \approx \mathbf{x}$, where the pseudo-inverse of \mathbf{y} is $\sim \mathbf{y} := (y_1, y_d, y_{d-1}, \dots, y_2)^\top$, which is also called an involution [64].

Circular convolution in the prior formulation takes $O(n^2)$ time to compute, but an important factor in the practical application of HRRs is that circular convolution can be computed in $O(n \log n)$ using Fast Fourier Transforms (FFTs):

$$\mathbf{x} \circledast \mathbf{y} = \mathcal{F}^{-1}\{\mathcal{F}\{\mathbf{x}\} \odot \mathcal{F}\{\mathbf{y}\}\},$$

where \mathcal{F} is the discrete Fourier Transform operator and \odot is element-wise multiplication of complex vectors [65]. For circular convolution, the approximate inverse of \mathbf{x} can also be calculated by taking the complex conjugate of all its Fourier coefficients.

Key properties of circular convolution are that the operation is associative and commutative [64]. This means that $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^d$, $\mathbf{x} \circledast (\mathbf{y} \circledast \mathbf{z}) = (\mathbf{x} \circledast \mathbf{y}) \circledast \mathbf{z}$ and $\mathbf{x} \circledast \mathbf{y} = \mathbf{y} \circledast \mathbf{x}$. These properties make an HRR VSA a good choice for binding slot-filler pairs together where the order does not matter.

Plate refers to unitary vectors in an HRR VSA as the set of vectors with Fourier coefficients that all have a magnitude of 1. This results in the following properties: 1) the approximate inverse of a unitary vector is equal to its exact inverse; 2) the L2-norm of a unitary vector is 1; 3) binding two unitary vectors results in another unitary vector [65]. Thus, if one wanted to turn arbitrary vectors into unitary vectors in HRR VSA, then one would divide each of its Fourier coefficients by their magnitudes.

As an example of the common slot-filler pair use for HRR VSA, consider the sentence “The hairy dog licked the human”. To represent the sentence, one could define the following set of randomly initialized vectors to represent the potential roles of the sentence: **object**, **subject**, **verb**, **adjective**, and another set to represent the fillers: **hairy**, **dog**, **licked**,

human. The VSA vector to represent the entire sentence would then be:

$$\mathbf{sentence} = \mathbf{object} \otimes (\mathbf{adjective} \otimes \mathbf{hairy} + \mathbf{object} \otimes \mathbf{dog}) + \mathbf{verb} \otimes \mathbf{licked} + \mathbf{subject} \otimes \mathbf{human}.$$

Assuming a high enough dimensionality, all the slot and filler vectors should be dissimilar enough to distinguish between them, and also any bound pair of them, e.g., $\mathbf{verb} \otimes \mathbf{licked}$ should be unlike any other vector. Furthermore, one can query different roles in the sentence by binding the approximate inverse, i.e. involution, of the desired role vector with **sentence**:

$$\begin{aligned} \sim \mathbf{subject} \otimes \mathbf{sentence} &= \sim \mathbf{subject} \otimes (\mathbf{object} \otimes (\mathbf{adjective} \otimes \mathbf{hairy} + \mathbf{object} \otimes \mathbf{dog}) \\ &\quad + \mathbf{verb} \otimes \mathbf{licked} + \mathbf{subject} \otimes \mathbf{human}) \\ &= \sim \mathbf{subject} \otimes \mathbf{object} \otimes (\mathbf{adjective} \otimes \mathbf{hairy} + \mathbf{object} \otimes \mathbf{dog}) \\ &\quad \sim \mathbf{subject} \otimes \mathbf{verb} \otimes \mathbf{licked} + \sim \mathbf{subject} \otimes \mathbf{subject} \otimes \mathbf{human} \\ &= \mathit{noise} + \mathit{noise} + \mathbf{human} \\ &\approx \mathbf{human} \end{aligned}$$

The query above is asking what the subject is in the sentence. Since the algebra is distributive, $\sim \mathbf{subject}$ is applied to each term that is added together. $\sim \mathbf{subject} \otimes \mathbf{subject}$ results in the identity vector which makes the last term in the sentence equivalent to the answer of the query, **human**. The computation above treats $\sim \mathbf{subject}$ bound with any other vector in the VSA as noise, which is what happens in the first two terms that do not have **subject**. The actual result of the following binding, $\sim \mathbf{subject} \otimes \mathbf{verb} \otimes \mathbf{licked}$, is still a vector in the vector space, however, the result will be quasi-orthogonal to the vectors in our vocabulary due to the high dimensionality of the vector space. Since addition of vectors is similar to both of the operands, the expanded form of the equation above suggests $\sim \mathbf{subject} \otimes \mathbf{sentence}$ should have high similarity to **human**, $\sim \mathbf{subject} \otimes \mathbf{verb} \otimes \mathbf{licked}$, and to $\sim \mathbf{subject} \otimes \mathbf{object} \otimes (\dots)$. However, $\sim \mathbf{subject} \otimes \mathbf{verb} \otimes \mathbf{licked}$ and $\sim \mathbf{subject} \otimes \mathbf{object} \otimes (\dots)$ are not in our vocabulary, so we can treat those vectors as noise relative to the vectors that are in our vocabulary. To verify the answer, one can do a clean-up memory operation, where the vector in the vocabulary that has the highest similarity to the result of $\sim \mathbf{subject} \otimes \mathbf{sentence}$ is the answer of the query.

One could also do a nested query, where one could query for the object of the sentence and also the adjective of that object:

$$\begin{aligned}
& \sim\text{adjective} \otimes \sim\text{object} \otimes \text{sentence} \\
& = \sim\text{adjective} \otimes \sim\text{object} \otimes ((\text{object} \otimes (\text{adjective} \otimes \text{hairy} + \text{object} \otimes \text{dog}) \\
& \quad + \text{verb} \otimes \text{licked} + \text{subject} \otimes \text{human}) + \text{verb} \otimes \text{licked} + \text{subject} \otimes \text{human}) \\
& = \sim\text{adjective} \otimes (\text{adjective} \otimes \text{hairy} + \text{object} \otimes \text{dog}) + \\
& \quad \sim\text{adjective} \otimes \sim\text{object} \otimes \text{verb} \otimes \text{licked} + \sim\text{adjective} \otimes \sim\text{object} \otimes \text{subject} \otimes \text{human} \\
& = \text{hairy} + \sim\text{adjective} \otimes \text{object} \otimes \text{dog} + \text{noise} + \text{noise} \\
& \approx \text{hairy} + \text{noise} + \text{noise} + \text{noise} \\
& \approx \text{hairy}
\end{aligned}$$

Performing a clean-up memory on the result of this query would correctly return the word “hairy” as a response to “What is the adjective describing the object of the sentence?”.

Crawford et al. [13] used the HRR VSA with 512-dimensional vectors to encode WordNet, a human-scale knowledge base, and were able to extract entities using a similar method as shown above with an accuracy of close to 100%. Furthermore, these representations and processing were also implemented in a biologically plausible manner using a spiking neural network.

2.3.3 Vector-Derived Transformation Bindings (VTB)

Like HRRs, the bundling operation for VTBs is element-wise addition [32]. In contrast to HRRs, VTBs have a non-associative and non-commutative binding operation defined on vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. Specifically, given $d^{\frac{1}{2}} = d' \in \mathbb{N}_{>0}$, we have

$$\mathbf{x} \otimes \mathbf{y} := \mathbf{V}_y \mathbf{x} = \begin{bmatrix} \mathbf{V}'_y & 0 & 0 \\ 0 & \mathbf{V}'_y & 0 \\ 0 & 0 & \ddots \end{bmatrix} \mathbf{x}$$

, where

$$\mathbf{V}'_y = d^{\frac{1}{4}} \begin{bmatrix} y_1 & y_2 & \cdots & y_{d'} \\ y_{d'+1} & y_{d'+2} & \cdots & y_{2d'} \\ \vdots & \vdots & \ddots & \vdots \\ y_{d-d'+1} & y_{d-d'+2} & \cdots & y_d \end{bmatrix}.$$

The approximate inverse to \mathbf{y} is $\sim\mathbf{y}$, where the elements of \mathbf{y} are permuted such that $\mathbf{V}_{\sim\mathbf{y}} = \mathbf{V}_{\mathbf{y}}^\top$. The unitary vectors in VTB are all vectors $u \in \mathbb{R}^d$ such that $\mathbf{V}_{\mathbf{u}}$ is orthogonal.

The VTB binding operation has only right inverses and identities, so there exists an alternative Transposed VTB (TVTB) algebra, with two-sided inverses and identities with the following binding operation:

$$\mathbf{x} \circledast \mathbf{y} := \mathbf{V}_{\mathbf{y}}^\top \mathbf{x} = \begin{bmatrix} \mathbf{V}'_{\mathbf{y}}{}^\top & 0 & 0 \\ 0 & \mathbf{V}'_{\mathbf{y}}{}^\top & 0 \\ 0 & 0 & \ddots \end{bmatrix} \mathbf{x}$$

where $\mathbf{V}'_{\mathbf{y}}$ is the same as in VTB.

To use the non-commutativity as an advantage, we can represent the relation between to entities with just two binding operations. If we want to encode the sentence “Kyle likes pasta” with the vectors **Kyle**, **likes**, **pasta**, we could try representing it simply with **sentence** = **Kyle** \circledast **likes** \circledast **pasta**. Given the non-commutative nature of the binding operation, this representation is structurally different than **pasta** \circledast **likes** \circledast **Kyle**. However, querying this information is more difficult since we have to be specific about the order that we successively bind the vectors due to non-associativity and also specific about which side of **sentence** we apply the unbinding to, because of non-commutativity. If **Kyle** \circledast **likes** is bound, then **pasta** is bound to that result (i.e., **(Kyle** \circledast **likes)** \circledast **pasta**), then we can make the query “Who/what likes pasta?” with the following:

$$\begin{aligned} (\mathbf{sentence} \circledast \sim\mathbf{pasta}) \circledast \sim\mathbf{likes} &= ((\mathbf{Kyle} \circledast \mathbf{likes}) \circledast \mathbf{pasta} \circledast \sim\mathbf{pasta}) \circledast \sim\mathbf{likes} \\ &\approx \mathbf{Kyle} \circledast \mathbf{likes} \circledast \sim\mathbf{likes} \\ &\approx \mathbf{Kyle} \end{aligned}$$

However, if **sentence** = **Kyle** \circledast (**likes** \circledast **pasta**), then making the same query would be different. Specifically, it would be:

$$\begin{aligned} \mathbf{sentence} \circledast \sim(\mathbf{pasta} \circledast \mathbf{likes}) &= \mathbf{Kyle} \circledast (\mathbf{likes} \circledast \mathbf{pasta}) \circledast \sim(\mathbf{pasta} \circledast \mathbf{likes}) \\ &\approx \mathbf{Kyle} \end{aligned}$$

With this simple example, it evident that querying VTB VSA vectors is more involved than with HRR, but if directionality in vector associations is important, VTB is a sensible VSA option.

The encoding capacity of VTB is similar to HRR for flat, list structures, i.e. ones that do not have many successive bindings, but may have many additions. For stack encoding, i.e. many recursively bound items, VTB outperforms HRR in its retrieval capacity as the depth of the stack increases. For an encoded stack of 7 vectors, the minimum similarity any of the unbound (retrieved) vectors have with their original vector is close to 0.2 for VTB, but is closer to 0.05 for HRR [32].

2.3.4 Use of VSAs with LLMs For Question Answering

As discussed in Section 2.2, knowledge graphs have been a popular external knowledge base to augment language models in tasks like question-answering. The VSAs introduced in this chapter are designed to represent structured knowledge, such as the structure found in knowledge graphs, and they also scale well with an increase in the number of symbols and relations to be represented and computed over, due to the vector-dimensionality-preserving properties of VSA operations. This strength of VSA-based representations gave rise to the VSA encodings of graphs proposed in Section 3.1.1, which are then compared to an equivalent GNN-based MCQA model, QA-GNN, in Chapter 4.

Chapter 3

Methodology

In this chapter, we discuss the architecture of QAVSA in Section 3.1, the datasets and knowledge graphs used for experimentation in Section 3.2, and the graph data pre-processing and encoding for QAVSA in Sections 3.3 and 3.4. We then describe the different variations of QAVSA tested and the hyperparameter training details in Sections 3.5 and 3.6. The final hyperparameters of QAVSA used for the experimental results presented in Chapter 4 are listed in Section 3.7. Portions of Sections 3.1, 3.2, 3.3, and 3.4 contain content from a previously published paper [50].

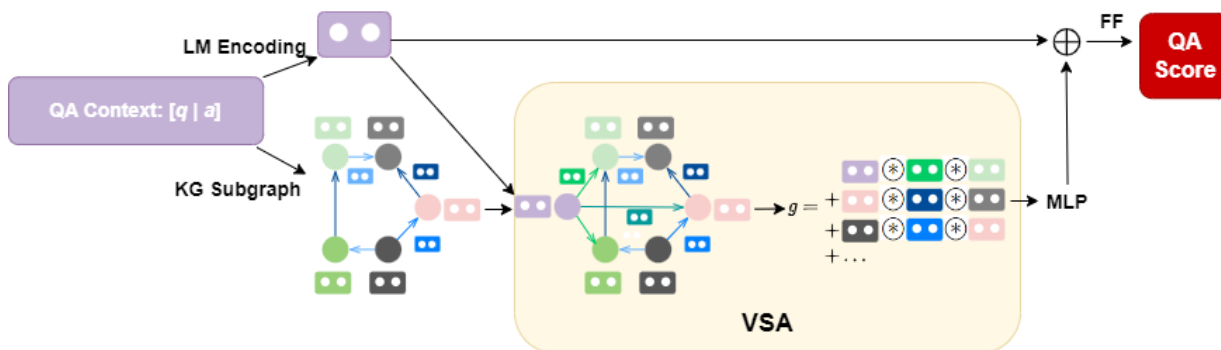


Figure 3.1: QAVSA model outline. The QA context, $[q|a]$, is input to a PLM to generate an LM encoding for the context and is also used to generate a KG subgraph. The LM encoded QA context is added to the graph, and the graph is converted to a single vector using a VSA. The VSA representation is fed through a k-layer MLP, concatenated with the original QA context encoding, and passed through a single FF layer to score the QA pair.

3.1 QAVSA Architecture

Given a multiple-choice question q and an answer option a , as in QA-GNN [95], the purpose of the QAVSA model is to score the plausibility of each (q, a) pair from the set of all answer options by performing joint reasoning using a (q, a) context, \mathbf{z} , generated from a pretrained language model (PLM) encoder, and a working graph G_w that contains relations and concepts pertaining to each specific (q, a) pair. In QA-GNN, the graph reasoning portion of the model consists of a specific type of GNN called a Graph Attention Network (GAT; [83]). As a replacement for the GAT in QAVSA, a single VSA vector representation of the (q, a) graph is generated. This representation is used as input to a simple k -layer MLP to realize the graph reasoning portion of QAVSA. Using this learned MLP along with the PLM (q, a) context embeddings, the (q, a) pair is scored, as shown in Figure 3.1.

3.1.1 Vector Symbolic Algebra Representation of the Knowledge Graph

Given the working graph for a (q, a) pair, $G_w = (V_w, E_w)$, and initial concept and relation embeddings $\mathbf{v}_i, i \in [0, 1, \dots, |V_w|]$, $\mathbf{r}_j, j \in [0, 1, \dots, |R|]$, the graph can be broken into a set of subgraphs, each of which has a single head and tail connected by an edge. The VSA representation of a given head-edge-tail triple can be computed as follows. For triple $triple_k = (h_k, e_k, t_k)$, $k \in [0, 1, \dots, |E_w|]$, where h_k, e_k, t_k specifies the type of entity for the head, relation, and tail of the triple, respectively, using the methods of Section 2.3 we bind each of the elements together using the binding operation of the given VSA: $\mathbf{triple}_k = \mathbf{v}_{h_k} \otimes \mathbf{r}_{e_k} \otimes \mathbf{v}_{t_k}$.

For a concept $v_i \in V_w$ with initial vector representation \mathbf{v}_i , a structured representation of the concept can be calculated using its relations to other concepts in the working graph, G_w . Let T_i be the set of all triples in G_w that have v_i as the head entity. The VSA representation for v_i , $\mathbf{v}_{i_{vsa}}$, contains contextual relational information from the working subgraph since it is the bundling of all the triple VSA representations v_i appears in. $\mathbf{v}_{i_{vsa}}$ can be calculated as $\mathbf{v}_{i_{vsa}} = \sum_{k=1}^{|T_i|} \mathbf{triple}_k$, for $triple_k \in T_i$.

The working graph VSA representation can be calculated by adding up all the structured representations of each concept in the graph, which is also equivalent to the sum of all the triples in the working graph:

$$\mathbf{g}_{vsa} = \sum_{i=1}^{|V_w|} \mathbf{v}_{i_{vsa}} = \sum_{k=1}^{|E_w|} \mathbf{v}_{h_k} \otimes \mathbf{r}_{e_k} \otimes \mathbf{v}_{t_k}.$$

Since circular convolution is commutative, \mathbf{triple}_k does not contain information on the direction of the relation, so a specific permutation σ can be applied to either the head or tail element of each triple to specify the directionality of the relation. To query a triple for a permuted concept, σ^{-1} is applied after unbinding.

As shown in Figure 3.1, given a QA input for question q and answer option a , an LM representation of the context is generated with a pretrained encoder to generate $LM(q|a) = \mathbf{z}$. We can also integrate \mathbf{z} into \mathbf{g}_{vsa} , analogous to [95], by forming new triples that bind \mathbf{z} with two new defined relation vectors, **IsAnswerConcept** and **IsQuestionConcept**, along with the corresponding answer and question concepts in \mathbf{g}_{vsa} . These triples are then added to \mathbf{g}_{vsa} as usual.

For example, the question in the CSQA dataset “What is the primary purpose of cars?” has the answer options {cost money, slow down, move people, turn right}, with the correct answer being “move people”. The subgraph for the QA context [What is the primary purpose of cars? move people] has question concepts $V_q = \{\text{PURPOSE, CAR, PRIMARY, CARS}\}$, answer concepts $V_a = \{\text{PEOPLE, MOVE, MOVE_PEOPLE}\}$, and many intermediate concepts, along with a set of triples, $E = \{(\text{MOVE, ANTONYM, STOP}), (\text{CAR, CAPABLE_OF, GO_FAST}), \dots\}$. The graph VSA vector is computed as

$$\begin{aligned} \mathbf{g}_{\text{vsa}} = & (\text{MOVE} \otimes \text{ANTONYM} \otimes \text{STOP})_{\text{vsa}} \\ & + (\text{CAR} \otimes \text{CAPABLE_OF} \otimes \text{GO_FAST})_{\text{vsa}} + \dots \end{aligned}$$

\mathbf{g}_{vsa} is then used as the input to a k -layer MLP with dropout and layer normalization, $MLP(\mathbf{g}_{\text{vsa}}) = \mathbf{g}_{\text{vsa}}^*$, and is responsible for learning to update the VSA representations within the graph vectors to solve the task (see Figure 3.1).

A plausibility score, i.e. the probability of answer a being correct, is computed with $p(a|q) \propto \exp(FF(\mathbf{z} \oplus \mathbf{g}_{\text{vsa}}^*))$, where the initial QA context \mathbf{z} is concatenated with the final graph VSA representation, $\mathbf{g}_{\text{vsa}}^*$, and is passed through a final feedforward layer.

3.1.2 PLM Encoding of the QA-context

All of the PLMs used to encode the QA-contexts are BERT-based, which all take the same form of input. Each input string has the form $s = \{[CLS], t_0, t_1, \dots, t_N, [SEP]\}$, where N is the maximum sequence length of the model, as mentioned in Section 2.1.3. The QA-context encoding, $LM(q|a) = \mathbf{z}$, is actually the pooled output of the last layer of the

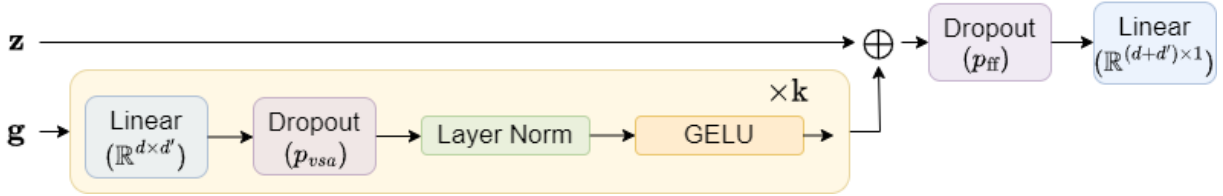


Figure 3.2: The feedforward layer architecture in QAVSA with the graph VSA vector, \mathbf{g} , and LM QA-context, \mathbf{z} , as input. The dimensionality of the weight matrices in the Linear modules are shown, and the probability of dropout is shown with the Dropout modules.

BERT-based PLM hidden states. The pooler consists of a linear layer followed by a tanh activation function, and it takes only the d -dimensional hidden state of the $[CLS]$ token as input.

3.1.3 Feed Forward Layers

A standard Multi-layer Perceptron (MLP) is responsible for transforming the graph VSA representation as shown in Fig. 3.1. Each layer in the MLP has the following components: a Linear module, a Dropout module, a Layer Normalization module, and an activation function, as shown in Figure 3.2. The Linear module (`nn.Linear` in PyTorch) specifies the standard weight matrix and bias vector to be learned in a neural network layer. Following the Linear module, the Dropout module (`nn.Dropout` in PyTorch) randomly sets values in its input vector to 0 with probability p_{vsa} with each forward pass during training in order to prevent over-fitting. The Layer Normalization module (`nn.LayerNorm` in PyTorch) then applies the following formula to the mini-batch output of the Dropout module:

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta$$

where x is the input mini-batch, ϵ is a small constant, and γ and β are learnable parameters. The Gaussian Error Linear Unit (GELU) [34] is used as the activation function following layer normalization, which uses the Cumulative Distribution Function for a Gaussian Distribution, but can be approximated with

$$GELU(x) = 0.5x * (1 + \text{Tanh}(\sqrt{2/\pi} * (x + 0.044715 * x^3))).$$

After the output of the MLP block and the PLM encoding of the QA context are concatenated, another Dropout module applies dropout with probability p_{ff} . Given that the output of the MLP block has d' neurons, then the final feed-forward (FF) layer has $d + d'$ (1024+1024) neurons in the input layer and 1 neuron in the output layer. There is no dropout or normalization applied to the final layer, which outputs the score, or logit, for the (q, a) input, as shown in Figure 3.2.

3.2 Datasets and Knowledge Graphs

In order to perform a comparison to QA-GNN, we evaluate QAVSA on the same three datasets that were used to evaluate QA-GNN in [95]. However, we hyperparameter tune our model rather than keeping parameters the same as those used in QA-GNN to maximize accuracy on the benchmark development splits. The three datasets we use are CommonsenseQA, OpenbookQA, and MedQA-USMLE.

CommonsenseQA (CSQA) [78] is a 5-way multiple choice commonsense reasoning task that requires different types of commonsense knowledge. The dataset has 12,102 questions, and the in-house train/dev/test split is adapted from Lin et al. [53] as 8500/1221/1241.

OpenbookQA (OBQA) [60] is a 4-way multiple choice dataset aiming to assess human understanding of elementary-level science questions in an open-book setting. The dataset consists of a list of 1326 science facts along with 5957 elementary school science questions with a train/dev/test split of 4957/500/500.

MedQA-USMLE (MedQA) [39] is a 4-way multiple choice dataset based on questions from the United States Medical License Exams (USMLE). The English version of the dataset has 12,723 questions, with a train/dev/test split of 10178/1272/1273.

For CSQA and OBQA, the external KG used is ConceptNet [74]. ConceptNet consists of 799,273 common words or phrases connected by edges of 17 different merged relation types, after pre-processing. The method to initialize the concept and relation embeddings for ConceptNet are also described in Section 3.4, and uses PLMs BERT-large or RoBERTa-Large. Following Yasunaga et al. [95], RoBERTa-Large is the PLM used to encode the QA contexts in QAVSA for CSQA, and AristoRoBERTa [12] is used for the QA contexts for OBQA.

The external KG used for MedQA is the Unified Medical Language System (UMLS; [7]), a popular biomedical knowledge base with 300K nodes, 1M edges, and 98 relation types. Following Yasunaga et al. [93], the PLM encoder used to generate concept and relation

vector embeddings is BioLinkBERT, which is a specific version of LinkBERT that utilizes medical document hyperlinks. BioLinkBERT is pretrained on PubMed with citation links to perform both masked language modeling and document relation prediction [94].

Dataset	Question	Answers
CSQA	A revolving door is convenient for two direction travel, but it also serves as a security measure at a what?	A. bank B. library C. department store D. mall E. new york
OBQA	Which of these animal’s habitat is peat likely to be found?	A. dog B. cat C. human D. alligator
MedQA	A 61-year-old woman is brought to the emergency department because of crushing substernal chest pain at rest for the past 2 hours. She is diaphoretic. Her temperature is 37.5°C (99.5°F), pulse is 110/min, respirations are 21/min, and blood pressure is 115/65 mm Hg. An ECG shows ST elevation in I, aVL, and V2–V4. Coronary angiography shows an 80% stenosis in the left anterior descending artery. Which of the following is the most likely initial step in the pathogenesis of this patient’s coronary condition?	A. Intimal monocyte infiltration B. Platelet activation C. Endothelial cell dysfunction D. Fibrous plaque formation

Table 3.1: Example questions from the development split of CSQA, OBQA, and MedQA.

An example multiple choice question from each of the datasets is shown in Table 3.1. CSQA questions focus on common objects and situations that require some commonsense reasoning to answer. In the example provided, one requires the commonsense knowledge of where revolving doors are usually located and also where security measures are necessary. OBQA, on the other hand, has questions limited to elementary-level science questions that are generally also common knowledge. MedQA is the hardest of the three datasets due to the fact that the questions do not test common knowledge, but test very specialized knowledge that only doctors would be expected to know how to answer. Most of the questions involve diagnosing or treating hypothetical patients, as can be seen in the example provided.

OBQA differs from the other two datasets because it also provides a science fact sheet that are generally relevant to the dataset questions. Examples of such facts are listed in Table 3.2.

an example of a fossil is a footprint in a rock
adding salt to a solid decreases the freezing point of that solid
sugars are transported from the leaves to the roots of a plant
the respiratory system transfers oxygen to the circulatory system

Table 3.2: Example facts from the OBQA fact sheet.

3.3 Knowledge Graph Entity Embeddings

Feng et al. [21] provide BERT-based embeddings computed for each concept in the knowledge graphs used by QA-GNN, which the authors refer to as TZW embeddings. However, these embeddings do not include the relation embeddings of the knowledge graphs e.g., ANTONYM, CAPABLE_OF, RELATED_TO, etc. The code to exactly reproduce Feng’s embeddings is not included in the paper’s code repository nor do the authors of the paper have the code (we contacted Yanlin Feng by email who confirmed they do not have the desired code). The embeddings were contributed by a member outside of the author list, and no further contact information of this individual was provided by the authors. Since the original source code for the embeddings was unobtainable, we computed all embeddings, including the graph concepts and relations, following the process described in [21].

The initial 1024-dimensional embeddings of the concepts, i.e. graph nodes, and relations, i.e. graph edges, are defined by feeding each triple composed of a head, relation and tail entity, $(h, r, t) \in V_w \times R \times V_w$, as a sentence into the previously mentioned PLM text encoders. The representations for each entity, referring to either nodes or edges in the knowledge graph, are pooled by taking the average representation of the corresponding portion of each triple that they appear in. Let $\{h_0, h_1, \dots, h_H\}$, $\{r_0, r_1, \dots, r_R\}$, $\{t_0, t_1, \dots, t_T\}$ be the sequences of tokens for the head, relation, and tail entities, respectively. The triple sentence is composed as $\{h_0, \dots, h_H, r_0, \dots, r_R, t_0, \dots, t_T, \text{PAD}\}$, where PAD is padding that makes the input length equal to the maximum input length of the model. The embeddings for h , r , and t are initialized to $[0, 0, \dots] \in \mathbb{R}^{1024}$ and are updated by $h_{emb+} = \frac{\sum_{i=0}^H PLM(h_i)}{H}$, $r_{emb+} = \frac{\sum_{i=0}^R PLM(r_i)}{R}$, and $t_{emb+} = \frac{\sum_{i=0}^T PLM(t_i)}{T}$, respectively. After each entity embedding has been aggregated from every triple in the graph, they are divided by the number of triples they appear in.

Computing relation embeddings, r_{emb} , in this way was unnecessary for QA-GNN because they represent the relation type as a one-hot vector for their GNN module [95]. However, in our approach, each relation is a distributed representation, so we can use an initial vector embedding computed in the same way as each graph node embedding is

computed.

3.4 Knowledge Graph Pre-Processing

To generate the working graph G_w , we follow the exact pre-processing technique described in [53]. The external domain-specific or world knowledge relevant to the answer question task is defined by a knowledge graph $G = (V, E)$ made up of a set of nodes, V , and a set of directed edges to capture relations, $E \subseteq V \times R \times V$, connecting the nodes with relation types from the set R .

The nodes of the working subgraph G_w are selected by first linking the concepts from the question, V_q , and from the answer, V_a , to nodes in G , where $V_q \cup V_a = V_{q,a} \subset V$. All of the nodes on a 2-hop path between the nodes in $V_{q,a}$, i.e., all nodes in V related to two of the nodes in $V_{q,a}$ are also included in the working graph, producing V_w . Finally, V_w is pruned down to 200 nodes by calculating the probability of a PLM generating each node given the QA-context as text input, as described in [95]. All of the edges connecting each pair of nodes in V_w , defined as $E_w \subset E$, are included in the final working graph, $G_w = (V_w, E_w)$.

3.5 Model Variations

A multitude of model variations on different components of QAVSA were tested throughout training, and these are summarized in Table 3.3.

The main component in the original QA-GNN architecture that we test to be included or altered in the final version of QAVSA is adding QA-context into the QA graphs or not. Several MLP architecture strategies were also tested for QAVSA. Given graph embedding dimensionality d , the following methods are compared:

1. Every layer has d neurons
2. Every layer after the input layer has $2d$ neurons (input layer has d neurons)
3. Every layer after the input layer has $d/2$ neurons (input layer has d neurons)
4. Every hidden layer has $2d$ neurons (input and output layers have d neurons)
5. Every hidden layer has $d/2$ neurons (input and output layers have d neurons)

Model Variation	Options
Algebra	Type = {HRR, TVTB} Permutation = {none, head entity, tail entity}
Embedding Encoder	CSQA & OBQA = {BERT, RoBERTa} MedQA = {BioLinkBERT}
Graph Embeddings	Relations = {Random, PLM} Entity Normalization = {Concept, Relation, Both} Graph Normalization = {Yes, No}
VSA MLP Dimensions (Input → Hidden → Output)	$d \rightarrow d \rightarrow d$, $d \rightarrow 2d \rightarrow \begin{cases} 2d \\ d \end{cases}$, $d \rightarrow d/2 \rightarrow \begin{cases} d/2 \\ d \end{cases}$
QA Context Embedding	PLM Encoder = $\begin{cases} \text{BERT, RoBERTa} & \text{for CSQA, OBQA} \\ \text{BioLinkBERT} & \text{for MedQA} \end{cases}$ Node in QA graph = {Yes, No}

Table 3.3: Summary of the QAVSA model variations that were tested.

The different hidden layer strategies listed above are motivated by the fact that for many neural network architectures including standard Dense feedforward networks and CNNs, the reduction or expansion of the hidden-layer size affects the complexity of features or functions of the input data that are computed [18].

Different versions of the graph VSA generation and pre-processing described in Section 3.1.1 and Section 3.4 are also compared. The PLMs used to generate initial concept embeddings for ConceptNet are RoBERTa-large and BERT-large (uncased), which are compared to the TZW embeddings provided by Feng et al. [21]. Bert-large-cased and BioLinkBERT-large are tested for the initial UMLS concept embedding generation and MedQA QA-context embedding.

For CSQA, OBQA, and MedQA, the HRR (Section 2.3.2) and TVTB (Section 2.3.3) VSAs are compared as methods of generating graph VSA embeddings. We compare the generation of the ConceptNet subgraphs for OBQA in several ways:

1. Using randomly initialized relation vector embeddings versus generating relation vector embeddings with PLMs (same method as the concept vector embeddings)
2. Normalizing the concept and/or relation embeddings before binding the vectors in triples versus keeping the embeddings un-normalized

3. Normalizing the graph VSA embeddings before inputting them into QAVSA versus keeping the graph VSA magnitudes (so that graphs with more triples generally have a larger magnitude)
4. With the HRR VSA, permuting the tail or head embedding in each triple versus permuting none of the embeddings

Testing random relation embeddings and PLM generated embeddings allows us to test whether orthogonality between the relations is more important than the semantic information of the relations.

Furthermore, the normalization of the concepts and relations was computed in two ways. The first way is to divide by the standard L2 norm of the vectors, where for a vector $\mathbf{x} \in \mathbb{R}^d$,

$$|\mathbf{x}| = \sqrt{\sum_{i=1}^d x_i^2}.$$

This method of normalization does not result in a desired property with VSAs where binding two unit length vectors produced a unit length vector. The second way is to make the vectors unitary, where the norm of unitary vectors is preserved under binding (see Section 2.3.2). The random relation embeddings were generated to be unitary and also to have a maximum Cosine similarity of 0.3 from any other concept or relation.

Testing entity normalization, i.e., before the concepts and relations are bound together, lets us determine if the differences in magnitude generated by the PLM embeddings are more important than using a standard unit magnitude across all the vectors used in the VSA. On the other hand, testing graph normalization lets us determine if the number of triples within a graph is beneficial to the neural network training, since increasing the number of triples contained in a graph VSA embedding increases the magnitude of the vector.

Residual connections are introduced in the MLP block of QAVSA in three ways: a single connection from the input to the last layer of the MLP, connections between consecutive layers, and connections from the input to each layer. These strategies can be seen in Figure 3.3. Since the input graph VSA representations are structured, these residual connections are introduced with the conceptual idea of maintaining the initial structured representation throughout later layers in the network. Also, the general performance improvements seen in other neural networks with residual connections inspired their introduction in QAVSA [56].

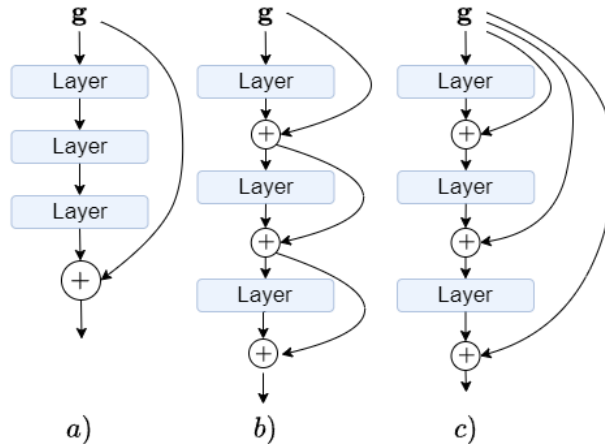


Figure 3.3: The three strategies for placing residual connections in QAVSA are: a) a single connection from the input graph vector, g , to last layer; b) connections between consecutive layers; c) connections from the input graph vector to each layer.

3.6 Training Details

The training loop for QAVSA is as follows. Each training epoch is divided into batches that are then further divided into mini-batches. Each sample of the data consists of a question, all of its multiple-choice answers, and a VSA subgraph for each question-answer pair. During training, the cross-entropy between the plausibility scores of all answer options are computed, and are back-propagated through both the LM encoder and VSA MLP components of the model. Given the set of question-answer pairs $A = a_0, a_1, \dots$ with binary labels $Y = y_0, y_1, \dots$, where $y_i = \begin{cases} 0 & \text{incorrect answer} \\ 1 & \text{correct answer} \end{cases}$, and logits outputted by QAVSA, $X = \{x_0, x_1, \dots\}$, the cross-entropy loss with Softmax applied to the logits is calculated as:

$$L = -\frac{1}{|A|} \sum_{a=1}^{|A|} \log\left(\frac{e^{x_a}}{\sum_{i=1}^{|A|} e^{x_i}}\right) \cdot y_a.$$

The CrossEntropyLoss function from the torch.nn Python package with a mean reduction was used to implement this loss. For each mini-batch, the average cross entropy loss divided by the size of the mini-batch is added to the running total loss. The accuracy of

Parameter	Values
k	{3, 5}
Encoder LR	$[1 \times 10^{-7}, 3 \times 10^{-5}]$
Decoder LR	$[1 \times 10^{-4}, 1 \times 10^{-2}]$
Warmup Steps	{0, 50, 100, 150}
Batch Size	{32, 64, 128}
Encoder Unfreeze Epoch	{1, 3, 5}
VSA MLP Dropout (p_{vsa})	{0.1, 0.2, ..., 0.7}
Concatenation Layer Dropout (p_{ff})	{0.1, 0.2, ..., 0.7}
QA-Context in QA Graph	{True, False}
Add QA-Context Transform	{True, False}

Table 3.4: Hyperparameter tuning on CSQA using Tree-structured Parzen Estimator as a sampler, median pruning rule with 20 startup trials and 6 warmup steps per trials. 195 trials were completed. QAVSA used L2-normalized embeddings, HRR VSA graphs with TZW concept embeddings, random relation embeddings and a linear decay LR schedule.

the model, calculated by $\frac{\# \text{ of correct predictions}}{m}$, is added to the running total loss and total accuracy of the model. Back-propagation is then called for each mini-batch. For each batch, the optimizer, which is RAdam [55] for all experiments, is updated along with the learning rate (LR) scheduler. After each epoch, the total training loss and accuracy of the model is recorded along with the loss and accuracy for the development (dev.) and test splits of the dataset.

Several rounds of hyperparameter tuning were conducted on the CSQA and OBQA datasets in order to improve model performance. QAVSA was initially hyperparameter tuned on CSQA, and the parameters that were varied are listed in Table 3.4. These hyperparameters include the number of layers in the MLP block, k; the learning rates of the PLM encoder and the rest of the feedforward layers, also referred to as the decoder; the number of warmup steps in the learning schedule; the batch size (bs); the epoch in which the PLM encoder weights are unfrozen; the dropout of the MLP layers; whether or not to include the QA-context in the QA graphs; and whether or not to include the extra linear layer applied to the QA-context. After 20 startup trials, a Tree-structured Parzen Estimator was used to sample new hyperparameters, and a median pruning rule was used to prune trials before they finished. All of the trials used the TZW embeddings provided by Feng et al. [21] and randomly initialized relation embeddings with the HRR VSA.

The best 7 trials in terms of best development (dev.) accuracy from the first hyperparameter tuning round had the following set of hyperparameter values: k = 3, bs = 64,

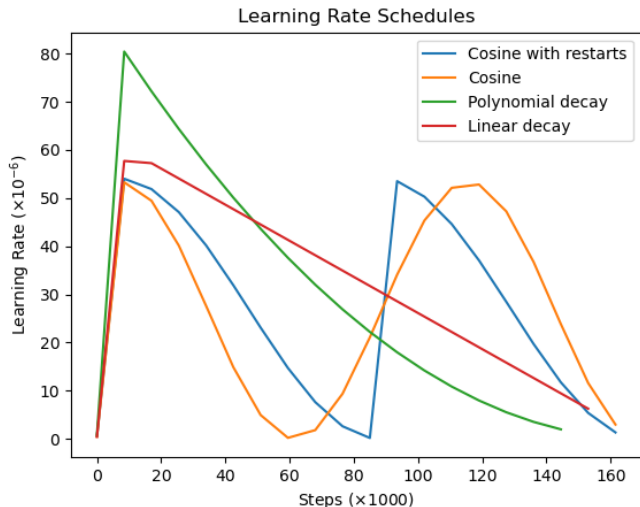


Figure 3.4: Examples of the different types of learning rate schedules tested. For these particular examples, the cosine and cosine with restarts schedules have 2 cycles, and the polynomial decay is quadratic (power of 2).

warmup = 100 steps, unfreeze epoch = 1 for the PLM weights, with QA-context added into the graphs but no QA-context transform layer. The best model has an accuracy of 93.12%, 77.97%, and 72.04% on the train, development, and test splits respectively. All of the encoder learning rates in the top trials were less than 1.55×10^{-5} with most of the learning rates around 1×10^{-5} , and the decoder learning rates were fairly variable within its range of $[1 \times 10^{-4}, 1 \times 10^{-2}]$. The dropout rate was the most variable, ranging from 0.1 to 0.5 for the VSA MLP dropout and from 0.3 to 0.7 for the concatenation layer dropout, however even with these variations, the top 20 trials all had dev. split accuracies above 76.90%. All of the top performing trials plateau in dev. and test accuracy after around 4 epochs, so the next goal was to slow down the learning rate and alter the learning scheduler.

The next round of hyperparameter tuning was also done on CSQA with the same sampler, pruner, and embeddings. The dropout is fixed at $p_{ff} = 0.3$ and $p_{vsa} = 0.1$. A particular emphasis on the learning rate schedule was tested in this round, and all the parameters are listed in Table 3.5. Examples of the LR schedules tested are plotted in Figure 3.4. The top trial in this round has an accuracy of 85.59%, 77.40%, and 71.07% on the train, development, and test splits respectively. Although the dev. accuracy is less than the previous round by 0.5%, there is less overfitting, given the smaller training accuracy. Out of the four learning rate schedules tested, the best trials used the Cosine

Parameter	Values
Encoder LR	$[1 \times 10^{-6}, 1.18 \times 10^{-5}]$
Decoder LR	$[5 \times 10^{-4}, 5.67 \times 10^{-3}]$
Warmup steps	{60, 80, 100, ..., 200}
LR Schedule	{Linear, Cosine, Cosine with restarts, Polynomial}
Polynomial Power	{2, 3, 4}
Cosine Cycles	{1, 2, 3}

Table 3.5: Hyperparameter tuning on CSQA using Tree-structured Parzen Estimator as a sampler, median pruning rule with 20 startup trials and 15 warmup steps per trials. 90 trials were completed. HRR VSA graphs with TZW concept embeddings and random relation embeddings were used.

learning rate with both 2 and 3 restarts. Also, the best trials had encoder learning rates less than 1×10^{-5} and decoder learning rates less than 5.6×10^{-3} . The new learning schedules delayed the best epoch to around two-thirds of the way through the training cycle.

The following round of hyperparameter tuning was also done on CSQA with the same sampler, no pruner, and with relation embeddings produced by BERT. The graph node (concept) embeddings are still the TZW embeddings provided by Feng et al. [21]. Additionally, we tested the performance of QAVSA without adding the QA-context into the QA graphs for all the trials, along with introducing the normalization of the input QA graphs and layer normalization in the VSA MLP layers. Overfitting was noticed in the previous round of tuning, so dropout was varied again in this round. The parameters are listed in Table 3.6. With this new configuration, the best trial has accuracies of 85.99%, 77.56%, and 71.64% on the train, dev., and test splits respectively. Layer normalization was used in some of the top trials, whereas graph normalization was not. The best trials had $k = 4$ instead of 3, PLM weight unfreezing at epoch 1, and had a mix of linear decay and cosine with restarts learning rates. Both p_{vsa} and p_{ff} ranged between 0.3 and 0.7.

The first hyperparameter tuning on OBQA was done with the same sampler, no pruner, and with our own computed BERT-generated concepts and relations rather than the TZW embeddings provided by Feng et al. [21]. The parameters are listed in Table 3.7. Compared to the previous tuning rounds on CSQA, the best trials in this OBQA tuning round all have an encoder unfreeze epoch of 4 instead of 1, $p_{vsa} = 0.4$, $p_{ff} = 0.2$, 2 cycles in the cosine learning rates schedule, a mix of $k = 3$ and 4, and a mix of including layer normalization and input graph normalization. Also, interestingly, none of the top trials add the QA context embedding into the QA graphs. The best trial has accuracies of 94.72%, 81.60%,

Parameter	Values
k	{2, 3, 4, 5}
Decoder LR	$[5 \times 10^{-6}, 1 \times 10^{-3}]$
Warmup steps	{0, 100, 200}
Encoder Unfreeze Epoch	{1, 5, 9, 13}
VSA MLP Dropout (p_{vsa})	{0.1, 0.2, ..., 0.7}
Concatenation Layer Dropout (p_{ff})	{0.1, 0.2, ..., 0.7}
LR Schedule	{Linear, Cosine, Cosine with restarts}
Normalize QA Graphs	{True, False}
Layer Normalization	{True, False}

Table 3.6: Hyperparameter tuning on CSQA using Tree-structured Parzen Estimator with 10 startup trials as a sampler and no pruning. 135 trials were completed. HRR CSA graphs with no QA-context node added, TZW concept embeddings, and BERT relation embeddings were used.

and 81.60% on the train, dev., and test splits respectively.

Hyperparameter tuning for MedQA was done in two stages. In both tuning rounds, the HRR VSA is used for encoding the graphs (with entity embeddings that are L2 normalized), QA contexts are not added to the graphs, normalization is not applied to the QA graph vectors, and layer normalization is applied in the VSA MLP block. To reduce the parameter search space, the first stage has the PLM weights frozen, and tests parameters like k , p_{vsa} , and the LR schedule. All the hyperparameters varied are listed in Table 3.8. From this first round, the best performing model parameters chosen for the second round were $k = 3$, a cosine learning schedule with 50 warmup steps, a decoder learning rate of 1.82×10^{-4} , a batch size of 32, and $p_{vsa} = 0.7$. The hyperparameters tested in the second round are ones involving the PLM, and are listed in Table 3.9. The top trials have encoder learning rates around 2×10^{-5} , $p_{ff} = 0.3$, and an unfreeze epoch of 6.

3.7 Final Hyperparameters

Table 3.10 lists the final QAVSA hyperparameters for each dataset. For all three datasets, the graph entity embeddings are normalized with the type of normalization listed under “Entity Norm”. Also, QAVSA has layer normalization in its MLP layers and no input graph normalization for all 3 datasets.

Parameter	Values
k	{3, 4, 5}
Encoder LR	$[1 \times 10^{-7}, 5 \times 10^{-4}]$
Decoder LR	[1e-5, 1e-2]
Encoder Unfreeze Epoch	{1, 4, 7}
VSA MLP Dropout (p_{vsa})	{0.2, 0.4, ..., 0.8}
Concatenation Layer Dropout (p_{ff})	{0.2, 0.4, ..., 0.8}
Cosine Cycles	{1, 2, 3}
Normalize QA Graphs	{True, False}
Layer Normalization	{True, False}
QA-Context in QA Graph	{True, False}

Table 3.7: Hyperparameter tuning on OBQA using Tree-structured Parzen Estimator with 10 startup trials as a sampler and no pruning with 150 trials completed. HRR VSA graphs with BERT concept and relation embeddings were used, and the LR schedule was Cosine w/ restart.

Parameter	Values
k	{3, 4, 5}
Decoder LR	$[1 \times 10^{-5}, 1 \times 10^{-2}]$
VSA MLP Dropout (p_{vsa})	{0.2, 0.3, ..., 0.7}
Batch Size	{32, 64, 128, 256}
Warmup steps	{0, 50, ..., 200}
LR Schedule	{Linear, Cosine with restarts}

Table 3.8: Hyperparameter tuning on MedQA with PLM weights frozen.

Parameter	Values
Encoder LR	$[1 \times 10^{-6}, 1 \times 10^{-4}]$
Encoder Unfreeze Epoch	{1, 6, 11}
Concatenation Layer Dropout (p_{ff})	{0, 0.1, ..., 0.5}

Table 3.9: Hyperparameter tuning on MedQA with unfrozen PLM weights.

Parameter	CSQA Value	OBQA Value	MedQA Value
VSA	HRR	HRR	HRR
Entity Norm	L2	Unitary	L2
k	5	4	3
Epochs	15	15	30
LR Schedule	cosine w/ restarts	cosine w/ restarts	cosine w/ restarts
LR Schedule Cycles	1	2	1
Warmup Steps	200	200	50
Batch Size	64	64	32
Encoder LR	1.77×10^{-5}	4.17×10^{-5}	2.07×10^{-5}
Decoder LR	3.71×10^{-2}	3.41×10^{-2}	1.82×10^{-4}
Encoder unfreeze epoch	4	4	6
VSA MLP Dropout (p_{vsa})	0.2	0.4	0.7
Final Layer Dropout (p_{ff})	0.4	0.2	0.3
Skip Connection	None	None	None

Table 3.10: The best model parameter values on all experiment datasets discovered by hyperparameter optimization.

Chapter 4

Results

In this chapter, we present the performance of QAVSA on two commonsense reasoning MCQA datasets, namely CSQA and OBQA, and on one medical MCQA dataset, MedQA. The hyperparameters of QAVSA on these datasets in Table 3.10. We display the accuracy of the dev. and test splits of these datasets are displayed in Section 4.1, along with the convergence rate of the model. In Section 4.2, we discuss the performance of variations and ablations of QAVSA, and in Section 4.3, we analyze the output embeddings of QAVSA. Some portions of this chapter include content from a previously published paper at the workshop on Representation Learning for NLP (RepL4NLP) [50].

4.1 Main Results

Model	IH Dev. Acc.	IH Test Acc. (%)
RoBERTa-Large* (w/o VSA KG)	75.23 (± 1.31)	69.14 (± 1.41)
+QA-GNN*	76.35 (± 0.95)	72.44 (± 1.87)
+QAVSA	76.76 (± 0.56)	70.83 (± 1.14)

Table 4.1: Accuracy and 95% confidence interval on CSQA inhouse dev. and test splits. Reproduced results (*) use reproduced node embeddings and all results are averaged over 5 different seeds.

As shown in Table 4.1, QAVSA has an improvement in mean accuracy of 0.41% and 1.53% compared to QA-GNN and RoBERTa-Large, respectively, on the CSQA inhouse dev.

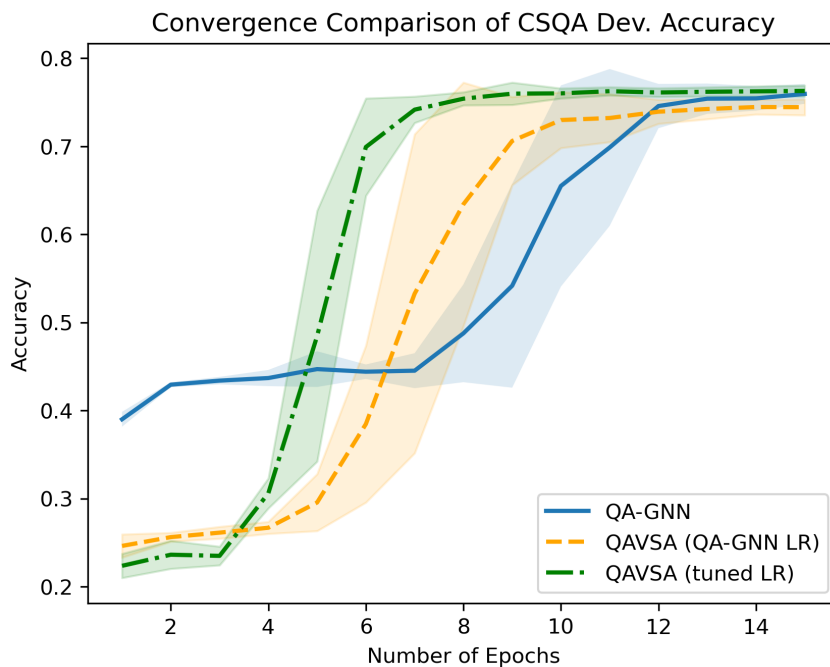


Figure 4.1: Mean accuracy of QA-GNN, QAVSA using our tuned LR schedule, and QAVSA using the LR schedule of QA-GNN on the CSQA dev. split for each epoch, averaged over 5 seed runs.

split. On the inhouse test split, QA-GNN outperforms QAVSA by a difference of 1.61% and improves upon the RoBERTa-Large baseline by 3.3%. However, based on the confidence intervals of the averaged seed runs, these difference are not statistically significant, except for the test split difference between QA-GNN and RoBERTa-Large.

On the OBQA dataset, QA-GNN has the best performance on the dev. split with a mean accuracy of 81.92%, as seen in Table 4.2. QAVSA is close behind with a mean accuracy of 81.72%, which is a 0.44% increase compared to the AristoRoBERTa baseline. However, on the test split QAVSA outperforms QA-GNN by a larger margin with a mean accuracy of 81.92%, which is a 1% improvement on the AristoRoBERTa baseline and a 1.56% improvement over QA-GNN. Again, the confidence intervals of the accuracies signify that the difference in accuracy is not statistically significant.

On the MedQA dataset, QAVSA outperforms the BioLinkBERT baseline and QA-GNN by 1.26% and 0.92%, respectively, on the dev. split, and by 0.54% and 0.4%, respectively, on the test split, as shown in Table 4.3. The performance difference between BioLinkBERT

Model	Dev. Acc. (%)	Test Acc. (%)
AristoRoBERTa* (w/o KG)	81.32 (± 0.76)	80.92 (± 0.83)
+QA-GNN*	81.92 (± 0.97)	80.36 (± 2.03)
+QAVSA	81.72 (± 1.41)	81.92 (± 0.99)

Table 4.2: Accuracy and 95% confidence interval on OBQA dev. and test splits. Reproduced results (*) use reproduced node embeddings and all results are averaged over 5 different seeds.

Model	Dev. Acc. (%)	Test Acc. (%)
BioLinkBERT* (w/o KG)	43.55 (± 0.20)	43.96 (± 0.30)
+QA-GNN*	43.89 (± 0.45)	44.10 (± 2.84)
+QAVSA	44.81 (± 0.78)	44.50 (± 1.87)

Table 4.3: Test accuracy and standard deviation on MedQA. Reproduced results are denoted with * and all results are averaged over 3 runs.

and QAVSA on the dev. split is statistically significant, based on the average performance of 3 seed runs.

In general, the reproduced results of QA-GNN on all the datasets is slightly lower than they report in their papers, and this could be due to an unnoticed difference in training environment, or due to the fact that the graph node embeddings of CPNet are different than the ones they used in their paper (see Section 3.3). However, it was not possible to reproduce their embeddings, so using those embeddings in QAVSA was not an option. We chose to use these embeddings to provide a fair comparison.

As shown in Figure 4.1, QAVSA converges faster during training. On average, it takes 12.4, 7.8, and 8.4 epochs to reach within 2.5% of each run’s maximum accuracy for QA-GNN, QAVSA (QA-GNN LR schedule), and QAVSA (our LR schedule), respectively. These convergence rates suggest that QAVSA can be trained 37% faster than QA-GNN, while performing with equivalent accuracy.

Although QAVSA does not consistently outperform QA-GNN, the architecture is much simpler than the Graph Attention Network in QA-GNN by only using standard MLP layers. There is no requirement to use node embedding matrices during computation along with linear and non-linear transformations on node and relation type embeddings to perform message passing between concepts, as is done in QA-GNN. Also, there is no requirement to use graph attention layers to create attention weights on the relations between concepts. In QAVSA, the attention on the relational edges within the graph arises naturally and can

be analyzed as shown in Section 4.2.

Additionally, the QAVSA memory requirements for its graph embeddings is constant at the dimensionality of the VSA vector, which in this case $d = 1024$. This compares favorably to GNNs that require an $N \times d$ node embedding matrix and an $N \times N$ adjacency matrix to represent a graph with N nodes. For these benchmarks, QA-GNN requires graphs with exactly 200 nodes for each QA pair. If one wanted to scale up the number of nodes in the graph significantly, the memory resources required would grow quadratically. In contrast, with an increase in the number of graph nodes and edges, the memory requirements for QAVSA are constant.

The simpler architecture of QAVSA, constant graph embeddings memory requirements, explainability that naturally arises from VSA-based encoding, and faster convergence demonstrate that our method should be preferred compared to the GNN-based method, QA-GNN.

4.2 QAVSA Variations and Ablations

Hidden Dim.	Output Dim.	Dev. Acc	Test Acc
512	512	75.10	69.38
	1024	75.35	68.41
2048	1024	75.10	68.82
	2048	75.59	70.51
1024	1024	77.23	71.96

Table 4.4: Dev. and test accuracy on CSQA with different VSA MLP dimensions. The input dimension for all trials is 1024.

The following results are for the variations and ablations of QAVSA that are described in Section 3.5.

Table 4.4 shows the result of varying the dimensions of the VSA MLP layers on the CSQA dataset. The best performance occurs when keeping the dimensionality of the MLP layers constant and equal to the VSA embedding dimensionality. As a note, Table 4.4 results are the output of a previous version of QAVSA with different hyperparameters than the final model version.

Table 4.5 displays the performance of QAVSA on CSQA and OBQA for CPNet graph entity embeddings generated by BERT and RoBERTa, and QA-contexts generated by the

Dataset	QA Context Emb.	Entity Emb.	
		BERT	RoBERTa
CSQA	BERT	62.57	61.67
	RoBERTa	76.08	75.68
OBQA	BERT	70.80	70.00
	RoBERTa	82.20	81.60

(a) Dev. acc (%)

Dataset	QA Context Emb.	Entity Emb.	
		BERT	RoBERTa
CSQA	BERT	56.49	55.44
	RoBERTa	70.83	71.96
OBQA	BERT	72.80	70.80
	RoBERTa	84.20	81.40

(b) Test. acc (%)

Table 4.5: Accuracy on CSQA and OBQA with different combinations of PLM QA-context encodings and graph concept/relation embeddings.

same PLMs. Using BERT as the PLM to generate the QA-contexts results in an over 10% drop in performance in every case, highlighting the improvements RoBERTa has over BERT as a sentence encoder. There is mostly no significant difference between graph entity (node and relation) embeddings generated by BERT or RoBERTa, with the differences mostly being less than 1%. However, for the majority of the cases, BERT does slightly outperform RoBERTa for graph entity embedding. This result is surprising due to the fact that RoBERTa generally outperforms BERT, as discussed in Section 2.1.3. The hyperparameter tuning of QAVSA is largely done using the BERT graph entity embeddings, and the slight gain in performance compared to RoBERTa suggests that the hyperparameters (i.e. learning rate or dropout rate) that maximize accuracy on these datasets with the BERT embeddings may differ from the ones that would maximize the performance of the RoBERTa graph entity embeddings. Another potential reason why RoBERTa does not outperform BERT in embedding is that the text sequences of graph triples used to compute these embeddings, as discussed in Section 3.3, are fairly short and direct, making the improvements in RoBERTa’s pretraining with dynamic masking and increased training length less relevant for this specific task.

A similar comparison between BioLinkBERT and Bert-Large (cased) for the graph entity encoder for the MedQA dataset is shown in Table 4.6. The largest performance

Split	QA Context Emb.	Entity Emb.	
		BERT	BioLinkBert
Dev.	BERT	30.98	31.37
	RoBERTa	30.58	29.01
	BioLinkBert	42.06	45.52
Test	BERT	31.66	31.50
	RoBERTa	30.79	23.25
	BioLinkBert	43.13	43.99

Table 4.6: Accuracy (%) of QAVSA on the dev. and test splits of MedQA with varied QA-context and knowledge graph entity embeddings.

difference occurs between BERT/RoBERTa versus BioLinkBert as a QA-context encoder, with differences in over 10% for both data splits. Unlike with CSQA and OBQA, BERT slightly outperforms RoBERTa as the QA-context encoder when using either BERT or BioLinkBERT to generate the UMLS graph entity embeddings. There is no significant difference between using BERT and BioLinkBERT as a graph entity encoder, with RoBERTa being an outlier with a large difference on the test split. Overall, the highest accuracy occurs when using BioLinkBert for encoding both the QA-contexts in MedQA and UMLS graph entities, which signifies that for MedQA, at least, the domain-specific pretraining of BioLinkBert is useful for encoding both sentences and graph entities in the medical domain.

Model Version	Dev. Acc. (%)	Test Acc. (%)
QAVSA	76.90	72.36
Unitary Norm	76.66	70.75
+ QA-context	76.09	71.23
+ Permutation (tail)	76.50	72.12
VTB	76.74	71.48
TVTB	75.76	70.51
+ Graph norm	75.84	70.75
+ Graph norm – Concept not norm	76.66	71.48
+ Skip 1	75.27	68.74
+ Skip 2	76.82	70.43
+ Skip 3	76.17	70.59
– BERT rels	76.17	71.88

Table 4.7: Accuracy of different model variations on dev. and test splits of CSQA.

Model Version	Dev. Acc. (%)	Test Acc. (%)
QAVSA	82.6	83.4
L2 Norm	80.60	81.00
+ QA-context	79.8	80.2
+ Permutation (head)	81.8	82.6
+ Permutation (tail)	81.6	83.2
VTB	81	80.2
TVTB	80.4	82.0
+ Graph norm	81.6	82.4
+ Graph norm – Concept not norm	83.0	82.8
+ Skip 1	81.60	84.20
+ Skip 2	80.40	82.80
+ Skip 3	82.00	82.40
– BERT rels	82.4	81.4
– Node Pruning	82	80.4

Table 4.8: Accuracy of different model variations on dev. and test splits of OBQA.

Results of several QAVSA model variations on the CSQA, OBQA, and MedQA benchmarks are shown in Tables 4.7, 4.8, and 4.9, respectively, with all other architecture parameters in Table 3.10 kept constant. The result in each of these tables corresponds to the results of the first QAVSA seed runs from Section 4.1. Including the QA context into the working graph drops the dev. and test accuracy by $\sim 3\%$ for OBQA and MedQA, and by $\sim 1\%$ for CSQA, suggesting that dynamically updating the graph representation in this way either muddles the original graph representation, or creates a VSA representation that is more difficult to learn by our current MLP architecture.

Introducing directionality in the VSA triples by means of permutation on the either the head or tail entities in the triple slightly drops accuracy for CSQA and OBQA, but has a more significant drop of $\sim 4\%$ on both splits of MedQA. This drop in performances indicates that binding through circular convolution stores enough semantic information from the graph for the task at hand.

The type of graph entity normalization creates slight performance changes, with the L2 norm outperforming unitary normalization by 1.61% on the test split of CSQA, and by $\sim 3\%$ on both splits of MedQA. For OBQA, QAVSA performs better with unitary normalization, outperforming L2 normalization by at least 2% for both splits. This difference between datasets suggests that the type of normalization should be determined on a case-by-case basis.

Model Version	Dev. Acc. (%)	Test Acc. (%)
QAVSA	45.52	43.99
Unitary Norm	41.90	41.32
+ QA-context	42.06	42.18
+ Permutation (head)	41.82	39.75
VTB + Graph Norm	39.07	40.69
TVTBT + Graph Norm	41.98	38.89
+ Graph norm	39.39	39.75
+ Skip 1	28.22	24.67
+ Skip 2	42.93	40.93
+ Skip 3	37.58	38.57

Table 4.9: Accuracy of different model variations on dev. and test splits of MedQA.

Furthermore, the HRR VSA is superior to other non-commutative algebras like VTB and TVTB with this architecture, with accuracy improvements on CSQA and OBQA of greater than 1% and up to 3.2%, compared to VTB and TVTB. Following a similar trend as previous results, the variation in accuracy is greater for MedQA, with differences between 4-6% between HRR and VTB/TVTBT.

Applying normalization to the graph VSA vectors drops performance more significantly when the concept vectors are also normalized, as can be seen for the CSQA and OBQA variations. For OBQA specifically, the dev. accuracy actually increases when normalizing the graph embeddings but not the concept embeddings. This indicates that including some type of information regarding the magnitude of either the graphs or concepts in the VSA vectors is useful for question answering. Not normalizing both graphs and concepts resulted in graph VSA vectors with too wide of a magnitude range for stable training. As a note, the VSA graph embeddings are normalized for VTB and TVTB for MedQA because otherwise, QAVSA does not improve above random chance after being trained. In general, the model variations for MedQA have much greater changes in accuracy. This variation indicates that QAVSA has less stability on this dataset, which is also supported by the fact that it is the hardest dataset compared to CSQA and OBQA, as evidenced by the greater than 30% accuracy difference of QAVSA on MedQA compared to the other two datasets.

Comparing the CPNet relation embeddings generated randomly versus those provided by BERT, performance drops with random embeddings by less than 1% for CSQA, and by 0.6% and 2% on the dev. and test splits of OBQA, respectively. Random relation embeddings have the benefit that they are more dissimilar to other relation and concept node embeddings compared to the PLM-generated embeddings, but have the downside of

not containing any intrinsic semantic content. The slight drop in performance with random relation embeddings suggests that the benefit of the inclusion of semantic information outweighs the higher similarity between relations and other concepts in the graph. On the other hand, the drop in performance is minimal, and this suggests that the structural symbolic content arising from the graph triple binding creates useful graph embeddings to compute over, even when the relations are randomly initialized. Also, as described in Section 3.4, the number of nodes in the QA subgraphs for all 3 datasets are normally pruned to 200 per QA pair. Using the entire grounded subgraph within CPNet for the OBQA QA graphs drops QAVSA accuracy by 3% on the test split, and this suggests that including the least relevant nodes to the QA pair in the graph does not add useful information for answering the question, and actually makes it more difficult to learn which relations in the graph are important. Furthermore, not pruning the subgraphs significantly increases the time to compute the graph VSA embeddings for each QA pair, especially when using the VTB or TVTB binding operators.

Adding skip connections in the 3 ways described in Section 3.5 decreases the accuracy more significantly on the test split of CSQA, and the most significantly on the dev. and test splits of MedQA. On the dev. split of CSQA and both splits of OBQA, adding in these skip connections does not change the accuracy of QAVSA too drastically, and the test accuracy even improves for the first skip connection method by 0.8% on the test split of OBQA.

4.3 Graph Embedding Analysis

We can analyze the effectiveness of the VSA MLP portion of QAVSA by computing the similarities of each triple VSA vector in a graph to the initial graph vector, \mathbf{g}_{vsa} , and the final graph vector, $\mathbf{g}_{\text{vsa}}^* = MLP(\mathbf{g}_{\text{vsa}})$ using the similarity operator defined in Section 2.3 and the MLP block defined in Figure 3.1. Such similarities can be used to determine which triples become the most prominent in the graph vector through the learned MLP transformation. Through this method of analyzing QAVSA output embeddings, we find that QAVSA does not always transform the input graph VSAs such that the output triples are always relevant to the reasoning required to answer the question. A consistent trend is that the most similar triples for each QA graph VSA embedding are ones with the most common relation type, which for CSQA and OBQA is RELATED_TO, and commonly appearing entities as the head or tail. Examples from the train and dev. splits of CSQA, OBQA, and MedQA are shown below to demonstrate 4 cases that can occur when looking at the most similar graph triples:

1. QAVSA predicts the correct answer and has relevant reasoning triples in the output graph embeddings.
2. QAVSA predicts the correct answer, but has irrelevant triples in the output graph embeddings.
3. QAVSA predicts the incorrect answer, but still has relevant triples, or sensible triples that may explain the incorrect answer
4. QAVSA predicts the incorrect answer and has irrelevant triples in the output graph embeddings.

To define some terminology used in the following examples, “initial triples” refers to the 20 triples with the highest similarity to \mathbf{g}_{vsa} , and “output triples” refers to the 20 triples with the highest similarity to $\mathbf{g}_{\text{vsa}}^*$. In cases where QAVSA predicts the correct answer, the QA graph corresponding to the correct answer is analyzed, and in cases where QAVSA predicts the incorrect answer, the QA graph of the prediction is analyzed along with the QA graph of the correct answer. With the incorrect predictions, the “initial triples” and “output triples” are both divided into “prediction triples” and “label triples” for the prediction answer option and correct answer option, respectively. Also, graph nodes and relations are italicized, with head-relation-tail triples notated with (*head, relation, tail*).

4.3.1 Case 1: Correct prediction and relevant triples

Starting with Case 1, the following question is from the train split of OBQA that QAVSA got correct: “Which object would have more tightly packed matter? **a) gold**, b) gas, c) wood, d) water”. The initial triples for the QA graph corresponding to the correct answer option (gold) all include the relation *related_to* and many of them have the entities *rock*, *gold*, and *metal*, like (*rock, related_to, metal*), (*metal, related_to, material*), and (*metal, related_to, solid*). However, with the output triples, they are more relevant to the reasoning required to answer the question, and there is more variety in the types of relations and concepts that appear. Specifically, there are triples that were not in the initial triples describing the “tightly packed” portion of the question like (*impaction, related_to, matter*), (*impaction, related_to, packed*), (*packed, related_to, jam-packed*), and (*jam-packed, related_to, tightly*), and other triples relating *gold* to “tightly packed matter” like (*impaction, related_to, mass*) and (*mass, related_to, gold*).

A simple example for Case 1 is a question from the dev. split of OBQA: “What is most necessary for a bitcoin operation? a) abacus, **b) metal**, c) plastic, d) wood”. Again,

the initial triples of the correct answer option (metal) includes triples such as (*operation, related_to, important*), (*operation, related_to, cut*), (*bit, related_to, metal*), and (*important, related_to, necessary*). A commonly appearing concept is *operation*, although *bit* and *metal* do appear in these triples. The most similar output triple is (*bit, related_to, bitcoin*), with (*bit, related_to, metal*) also appearing in the top 20 triples. These two triples demonstrate that QAVSA correctly learned to emphasize the direct connection between the question concept *bitcoin* and the correct answer *metal* in its output representation.

In the dev split of the CSQA dataset, an example for Case 1 is: “The end of the barrel of what primitive firearm is bell shaped? a) barbell, b) funnel, **c) blunderbuss**, d) wind instrument, e) kettlebell”. Initial triples for the correct answer (blunderbuss) include many triples with the concepts *shape*, *line*, and *cone*, like (*line, related_to, form*), (*cone, related_to, line*), and (*thimble, related_to, shape*), and blunderbuss does not appear at all. In the output triples, a single triple describes the answer to the question, (*bell, part_of, blunderbuss*), but other triples relating to guns also appear, like (*bullet, related_to, firearm*) and (*gun, related_to, rifle*). As a note, the other answer options are not even firearms, making this an easier question for a PLM to answer, even without the external KG, but the successful transformation of relevant graph triples is still validated through this method of analysis.

Successful examples like the previously described ones are rarer for the MedQA dataset, as the questions in MedQA are much more complex than CSQA and OBQA, and the entities within the UMLS KG are not standard like the ones in CPNet are (see Section 3.2). The following question is from the train split of MedQA: “A 12-year-old boy presents to the emergency department after falling from his bike. He is holding his right arm tenderly and complains of pain in his right wrist. When asked, he says that he fell after his front tire hit a rock and landed hard on his right hand. Upon physical examination he is found to have tenderness on the dorsal aspect of his wrist in between the extensor pollicis longus and the extensor pollicis brevis. Given this presentation, which of the following is the most likely bone to have been fractured? **a) Scaphoid**, b) Lunate, c) Pisiform, d) Capitate”. There are many initial triples relating to fractures, such as (*Secondary open reduction of fracture of bone, is a, Revision to open reduction of fracture and cast immobilization*) and (*Closed reduction of fracture and traction, is a, Primary closed reduction of fracture and skeletal traction*), but with no triples involving the location of the injury. However, the output triples include the actual answer entity, “Scaphoid”, and other triples relating to fractures and the wrist: (*Wrist joint structure, direct procedure site of, Repair of nonunion of scaphoid bone with radial styloidectomy*), (*Wrist joint structure, is a, Wrist region structure*), and (*Repair of fracture with sequestrectomy, is a, Operation on fracture*). Although the general content of the output triples relates to the question, the details of

wrist anatomy and other medical procedures are usually not relevant to the details of the question and are also more difficult to verify without a medical background.

4.3.2 Case 2: Correct prediction and irrelevant triples

Examples in Case 2 demonstrate the fact that QAVSA may not necessarily improve upon the baseline PLM consistently for every question. Commonly for MedQA, QAVSA predicts the correct answer even though the initial and processed triples have no relevance in answering the question. An example from the dev. split is: “A 10-year-old boy presents to the emergency department with his parents. The boy complains of fever, neck stiffness, and drowsiness for the last several days. His past medical history is noncontributory ... His heart rate is 100/min, respiratory rate is 22/min, blood pressure is 105/65 mm Hg, and temperature is 40.5°C (104.9°F) ... A lumbar puncture is performed and reveals the following: a) Binding with ergosterol in the cell membrane, **b) Inhibition of DNA polymerase**, c) Nucleoside reverse transcriptase inhibition, d) Cell wall synthesis inhibition”. The large majority of the initial triples describe different examinations or other facts that are irrelevant to the final question involving lumbar puncture, such as (*Abscess of brain, may cause, Drowsiness*) and (*Black water fever, may cause, Pyrexia of unknown origin*). Furthermore, most of the output triples are irrelevant to the question, like (*O/E - hyperpyrexia - greater than 40.5 degrees Celsius, interprets, Temperature*) and (*Radiation therapy procedure or service, is a, Therapeutic procedure*). Although some of the output triples line up with the facts of the case, like the temperature being 40.5 degrees Celsius, QAVSA seems unable to determine what triples in the graph, if any, are relevant to answering many medical case-style questions. This may be due to either missing information in the external KG, or the inherent complexity to medical-school level questions.

An easier example to analyze for Case 2 is a question from the dev. split of CSQA: “Reading newspaper one of many ways to practice your what? **a) literacy**, knowing how to read, money, buying, money bank”. QAVSA correctly predicted “literacy” as the answer to the question, but the initial triples are actually more relevant to the question than the output triples. The initial top triples include (*read, related_to, literate*), (*read, related_to, newspaper*), (*read, related_to, knowledge*), indicating that sometimes the most common concepts in the graph structure are also the ones that are relevant to answering the question. None of the output triples include the concept *newspaper* and fewer triples including READ than the initial triples, but has triples like (*learn_about_world, hassubevent, study*), (*book, related_to, school*), and (*primer, related_to, read*). In this example, the graph processing portion of QAVSA is doing the opposite of what would be useful to answer the

question, and this suggests the PLM used to encode the QA context seems to be doing most of the work in some cases when predicting the correct answer.

4.3.3 Case 3: Incorrect prediction and relevant triples

The following example for Case 3 is from the dev. split of CSQA: “Where are the most famous BBQ steakhouses in America? a) **texas**, b) building, c) kansas city, d) maine, e) falling down”. QAVSA predicted “kansas city” instead of the label “texas”, which is an understandable mistake considering both places are known for BBQ, and the external KG processing further validates the confusion. The initial label triples (for answer option “texas”) has only geographic triples like (*texas, part_of, united_states_of_america*) and (*united_states_of_america, related_to, america*), and the initial prediction triples (for “kansas city”) are similarly geographic, like (*city, related_to, town*), (*state, related_to, america*), and (*country, related_to, city*). However, the output triples differ more significantly between the answer options. The “texas” output triples include (*steakhouse, related_to, restaurant*) and (*heifer, at_location, texas*), along with triples relating to *america*, but has no triples directly relating *steakhouse* to *texas*. For “kansas city” however, the second most similar triple is (*steakhouse, at_location, kansas_city*), which directly connects the question asked to the answer concept. The reason QAVSA failed to answer this question correctly could be due to the ambiguity to which answer is more correct, but it could also suggest that future iterations of QAVSA could benefit from integrating the VSA-style embeddings and external knowledge processing layers with the PLM in more sophisticated ways.

A different type of outcome in Case 3 can be seen with an example in the dev. split of OBQA: “An example of data collection is: a) Deleting case files on the computer, b) Touching evidence without gloves, c) **speaking with a witness**, d) Throwing documents in the trash”. QAVSA incorrectly predicted b) as the answer, because in this example, both sets of initial triples and output triples have a similar amount of relevance to the question. The initial triples for both answers are not relevant to the question of data collection, but consist of triples relating to the answer concepts, like (*speak, related_to, conversation*) and (*voice, related_to, talking*) for “speaking with a witness” and (*feel, related_to, hand*) and (*feeling, related_to, touch*) for “Touching evidence without gloves”. The output triples for both options include (*traffic, related_to, information*), (*research, related_to, data_collection*), and (*statistic, is_a, datum*). Some differences are that the output label triples include (*knowledge, related_to, data*) and (*fact, related_to, knowledge*), whereas the output prediction triples include a reasoning path with (*touch, related_to, input*) and (*input, related_to, data*), not accounting for irrelevant triples to the question in both answer output embeddings. The reasoning path from *touch* to *data* is not as strong logically, but

it may have been a reason that b) was chosen over c). Another contributing factor to the error could be the overlap in the emphasis on triples involving data or data collection in the output, resulting in two final graph embeddings that are somewhat similar to one another.

4.3.4 Case 4: Incorrect prediction and irrelevant triples

Across the 3 datasets, there are a number of examples where QAVSA predicts the wrong answer and the most emphasized graph triples are nonsense. One such example is from the dev. split of CSQA: “If I was watching TV on the couch and the air was stuffy, I might turn the fan on to make the what more comfortable? a) hockey game, **b) living room**, c) bathroom, d) football stadium, e) sauna”. QAVSA predicted “sauna” instead of “living room”, and the initial triples for both include *(house, related_to, living_place)*, *(movement, related_to, move)*, and *(look, related_to, do)*. The output triples for both answer options are similar between each other as well, with most of them related to WATCH or other irrelevant concepts to the question, like *(watcher, related_to, watch)*, *(band, related_to, roll)*, *(see_favorite_show, has_subevent, watch_tv)*, and *(game, related_to, play)*. These errors suggest that there is room for improvement in QAVSA in terms of integrating and verifying the internal knowledge of PLMs with the verifiable external knowledge in the VSA embeddings. It may be useful to use the PLM sentence encodings of the QA-context to inform the graph VSA embeddings, either statically before fine-tuning or dynamically during fine-tuning, in ways other than adding the QA-context as a node in the graph.

Chapter 5

Conclusion

In this work, we have presented QAVSA, a new type of question answering, deep neural network model that leverages VSA-represented knowledge graphs along with general linguistic knowledge from PLMs to perform reasoning on MCQA benchmarks. Through a direct comparison to the GNN-based model QA-GNN on three benchmarks, namely CSQA, OBQA, and MedQA, we demonstrate that QAVSA performs on par with QA-GNN, while using a simpler k -layer MLP reasoning module (see Section 4.1). Furthermore, QAVSA exhibits better memory scaling than GNN-based models because of the constant memory requirements of our VSA-based graph representations (see Sections 2.3 and 3.1). We also demonstrate faster convergence during training than QA-GNN (see Section 4.1) and highlight the explainability of our model outputs by analyzing the output graph embeddings of QAVSA in cases where our model succeeds and fails (see Section 4.3).

Some limitations of QAVSA are highlighted in Section 4.3, where the analysis of the output embeddings does not always explain our model behavior. There are many examples, specifically in MedQA, where QAVSA is unable to learn the relations within graphs that contain highly domain-specific and complex concepts, as is the case with the UMLS knowledge graph. Also, even though QAVSA performs comparably to QA-GNN, there is not a significant improvement to the PLM baselines in some cases.

Future work can be done to improve the performance of QAVSA relative to the PLM baselines, such as using dynamic graph VSA embeddings that are updated during fine-tuning or using structural VSA embeddings as training objectives to make the PLM embeddings more structural during pretraining. Our model could also be applied to other MCQA datasets where external knowledge is relevant. Alternatively, one could adapt our method to question-answering systems that are not multiple-choice, which would be useful

for tasks that require step-by-step reasoning. GNNs are also widespread for tasks outside of natural language processing, such as object detection, chemical reaction prediction, and disease classification, and it is worthwhile to determine if our VSA-based approach is useful for representing structures that are not linguistic [101].

References

- [1] Rohan Anil, et al. PaLM 2 Technical Report, September 2023.
- [2] Rohan Anil, et al. Gemini: A Family of Highly Capable Multimodal Models, June 2024. arXiv:2312.11805.
- [3] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program Synthesis with Large Language Models, August 2021. arXiv:2108.07732.
- [4] Madeleine Bartlett, Terrence C. Stewart, and Jeff Orchard. Biologically-Based Neural Representations Enable Fast Online Shallow Reinforcement Learning. *Proceedings of the Annual Meeting of the Cognitive Science Society*, 44(44), 2022.
- [5] Madeline Bartlett, Katherine Simone, Nicole Dumont, P. Michael Furlong, Chris Eliasmith, Jeff Orchard, and Terrence C. Stewart. Improving Reinforcement Learning with Biologically Motivated Continuous State Representations. In *Proceedings of the International Conference on Cognitive Modeling*, 2023.
- [6] Peter Blouw, Eugene Solodkin, Paul Thagard, and Chris Eliasmith. Concepts as Semantic Pointers: A Framework and Computational Model. *Cognitive Science*, 40(5):1128–1162, 2016.
- [7] Olivier Bodenreider. The Unified Medical Language System (UMLS): integrating biomedical terminology. *Nucleic Acids Research*, 32(Database issue):D267–D270, January 2004.
- [8] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. A Survey on Evaluation of Large Language Models. *ACM Trans. Intell. Syst. Technol.*, 15(3):39:1–39:45, March 2024.

- [9] Mark Chen, et al. Evaluating Large Language Models Trained on Code, July 2021. arXiv:2107.03374.
- [10] Xuan Choo. *Spawn 2.0: Extending the World's Largest Functional Brain Model*. PhD thesis, UWSpace, 2018.
- [11] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge, March 2018. arXiv:1803.05457.
- [12] Peter Clark, Oren Etzioni, Daniel Khashabi, Tushar Khot, Bhavana Dalvi Mishra, Kyle Richardson, Ashish Sabharwal, Carissa Schoenick, Oyvind Tafjord, Niket Tandon, Sumithra Bhakthavatsalam, Dirk Groeneveld, Michal Guerquin, and Michael Schmitz. From F to A on the New York Regents Science Exams — An Overview of the Aristo Project. *AI Magazine*, 41(4):39–53, December 2020.
- [13] Eric Crawford, Matthew Gingerich, and Chris Eliasmith. Biologically plausible, human-scale knowledge representation. *Cognitive science*, 40(4):782–821, 2016. Publisher: Wiley Online Library.
- [14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.
- [15] Nicole Sandra-Yaffa Dumont and Chris Eliasmith. Accurate representation for spatial cognition using grid cells. In *42nd Annual Meeting of the Cognitive Science Society*, pages 2367–2373, Toronto, ON, 2020. Cognitive Science Society.
- [16] Nicole Sandra-Yaffa Dumont, P. Michael Furlong, Jeff Orchard, and Chris Eliasmith. Exploiting semantic information in a spiking neural SLAM system. *Frontiers in Neuroscience*, 17, 2023.
- [17] Nicole Sandra-Yaffa Dumont, Andreas Stöckel, P. Michael Furlong, Madeleine Bartlett, Chris Eliasmith, and Terrence C. Stewart. Biologically-Based Computation: How Neural Details and Dynamics Are Suited for Implementing a Variety of Algorithms. *Brain Sciences*, 13(2):245, January 2023. Publisher: MDPI AG.

- [18] Ronen Eldan and Ohad Shamir. The Power of Depth for Feedforward Neural Networks. *arXiv e-prints*, page arXiv:1512.03965, December 2015.
- [19] Chris Eliasmith. *How to Build a Brain: A Neural Architecture for Biological Cognition*. Oxford Series on Cognitive Models and Architectures. Oxford University Press, Oxford, New York, new to this edition:, new to this edition: edition, June 2015.
- [20] Chris Eliasmith, Terrence C. Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A Large-Scale Model of the Functioning Brain. *Science*, 338(6111):1202–1205, November 2012. Publisher: American Association for the Advancement of Science.
- [21] Yanlin Feng, Xinyue Chen, Bill Yuchen Lin, Peifeng Wang, Jun Yan, and Xiang Ren. Scalable Multi-Hop Relational Reasoning for Knowledge-Aware Question Answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1295–1309, Online, November 2020. Association for Computational Linguistics.
- [22] P. Michael Furlong and Chris Eliasmith. Fractional Binding in Vector Symbolic Architectures as Quasi-Probability Statements. In *44th Annual Meeting of the Cognitive Science Society*. Cognitive Science Society, 2022.
- [23] P. Michael Furlong and Chris Eliasmith. Bridging Cognitive Architectures and Generative Models with Vector Symbolic Algebras. *Proceedings of the AAAI Symposium Series*, 2(1):262–271, 2023.
- [24] P. Michael Furlong, Katherine Simone, Nicole Dumont, Madeline Bartlett, Terrence C. Stewart, Jeff Orchard, and Chris Eliasmith. Biologically-plausible Markov Chain Monte Carlo Sampling from Vector Symbolic Algebra-encoded Distributions. In *ICANN*, 2024.
- [25] P. Michael Furlong, Terrence C. Stewart, and Chris Eliasmith. Fractional Binding in Vector Symbolic Representations for Efficient Mutual Information Exploration. *ICRA Workshop: Towards Curious Robots: Modern Approaches for Intrinsically-Motivated Intelligent Behavior*, 2021.
- [26] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-Augmented Generation for Large Language Models: A Survey, March 2024. arXiv:2312.10997.

- [27] Ross Gayler. Multiplicative Binding, Representation Operators & Analogy. In *Advances in Analogy Research: Integration of Theory and Data from the Cognitive, Computational, and Neural Sciences*. 1998.
- [28] Ross W. Gayler. Vector Symbolic Architectures answer Jackendoff’s challenges for cognitive neuroscience. In Peter Slezak, editor, *Proceedings of the ICCS/ASCS Joint International Conference on Cognitive Science (ICCS/ASCS 2003)*, pages 133–138, Sydney, NSW, AU, July 2003. University of New South Wales. [eprint: 0412059v1](#).
- [29] Gaël Gendron, Qiming Bao, Michael Witbrock, and Gillian Dobbie. Large Language Models Are Not Strong Abstract Reasoners. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, volume 7, pages 6270–6278, August 2024. ISSN: 1045-0823.
- [30] A. N. Gorban and I. Y. Tyukin. Blessing of dimensionality: mathematical foundations of the statistical physics of data. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2118):20170237, March 2018. Publisher: Royal Society.
- [31] Jan Gosmann. *An Integrated Model of Context, Short-Term, and Long-Term Memory*. Ph.D. dissertation, University of Waterloo, Waterloo, ON, 2018.
- [32] Jan Gosmann and Chris Eliasmith. Vector-Derived Transformation Binding: An Improved Binding Operation for Deep Symbol-Like Processing in Neural Networks. *Neural Computation*, 31(5):849–869, May 2019.
- [33] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.
- [34] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs), June 2023. [arXiv:1606.08415](#).
- [35] Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large Language Models Can Self-Improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1051–1068, Singapore, December 2023. Association for Computational Linguistics.
- [36] Yongfeng Huang, Yanyang Li, Yichong Xu, Lin Zhang, Ruyi Gan, Jiaying Zhang, and Liwei Wang. MVP-Tuning: Multi-View Knowledge Retrieval with Prompt Tuning

- for Commonsense Reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13417–13432, Toronto, Canada, July 2023. Association for Computational Linguistics.
- [37] Ray Jackendoff. *Foundations of Language: Brain, Meaning, Grammar, Evolution*. Oxford University Press, January 2002.
- [38] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L elio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth ee Lacroix, and William El Sayed. Mistral 7B, October 2023. arXiv:2310.06825.
- [39] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What Disease Does This Patient Have? A Large-Scale Open Domain Question Answering Dataset from Medical Exams. *Applied Sciences*, 11(14):6421, January 2021.
- [40] Ivana Kaji c, Tobias Schr oder, Terrence C. Stewart, and Paul Thagard. The Semantic Pointer Theory of Emotion: Integrating Physiology, Appraisal, and Construction. *Cognitive Systems Research*, 2019.
- [41] Pentti Kanerva. The Spatter Code for Encoding Concepts at Many Levels. In Maria Marinaro and Pietro G. Morasso, editors, *ICANN '94*, pages 226–229, London, 1994. Springer.
- [42] Nora Kassner, Benno Krojer, and Hinrich Sch utze. Are Pretrained Language Models Symbolic Reasoners over Knowledge? In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 552–564, Online, November 2020. Association for Computational Linguistics.
- [43] Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. UNIFIEDQA: Crossing Format Boundaries with a Single QA System. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1896–1907, Online, November 2020. Association for Computational Linguistics.
- [44] Denis Kleyko, Dmitri Rachkovskij, Evgeny Osipov, and Abbas Rahimi. A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part II: Applications, Cognitive Models, and Challenges. *ACM Comput. Surv.*, 55(9):175:1–175:52, January 2023.

- [45] Denis Kleyko, Dmitri A. Rachkovskij, Evgeny Osipov, and Abbas Rahimi. A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part I: Models and Data Transformations. *ACM Comput. Surv.*, 55(6), December 2022. Place: New York, NY, USA Publisher: Association for Computing Machinery.
- [46] Brent Komer and Chris Eliasmith. Efficient navigation using a scalable, biologically inspired spatial representation. In *42nd Annual Meeting of the Cognitive Science Society*, Toronto, ON, 2020. Cognitive Science Society.
- [47] Brent Komer, Pawel Jaworski, Steven Harbour, Chris Eliasmith, and Travis DeWolf. BatSLAM: Neuromorphic Spatial Reasoning in 3D Environments. In *41st Digital Avionics Systems Conference*, Portsmouth, VA, USA, 2022. DASC.
- [48] Brent Komer, Terrence C. Stewart, Aaron R. Voelker, and Chris Eliasmith. A neural representation of continuous space using fractional binding. In *41st Annual Meeting of the Cognitive Science Society*, Montreal, QC, 2019. Cognitive Science Society.
- [49] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. A Survey on Complex Knowledge Base Question Answering: Methods, Challenges and Solutions. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 4483–4491, Montreal, Canada, August 2021. International Joint Conferences on Artificial Intelligence Organization.
- [50] Ryan Laube and Chris Eliasmith. QAVSA: Question Answering using Vector Symbolic Algebras. In *Proceedings of the 9th Workshop on Representation Learning for NLP (RepL4NLP-2024)*, pages 191–202, Bangkok, Thailand, August 2024. Association for Computational Linguistics.
- [51] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.
- [52] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Alexander Cosgrove, Christopher D. Manning, Christopher Re, Diana Acosta-Navas, Drew Arad Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao,

- Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Andrew Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic Evaluation of Language Models. *Transactions on Machine Learning Research*, February 2023.
- [53] Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. KagNet: Knowledge-Aware Graph Networks for Commonsense Reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2829–2839, Hong Kong, China, November 2019. Association for Computational Linguistics.
- [54] Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring How Models Mimic Human Falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland, May 2022. Association for Computational Linguistics.
- [55] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020.
- [56] Tianyi Liu, Minshuo Chen, Mo Zhou, Simon S. Du, Enlu Zhou, and Tuo Zhao. Towards understanding the importance of shortcut connections in residual networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 2019.
- [57] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, July 2019. arXiv:1907.11692.
- [58] Nicholas Lourie, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. UNICORN on RAINBOW: A Universal Commonsense Reasoning Model on a New Multitask Benchmark. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(15):13480–13488, May 2021.
- [59] Thomas Lu, Aaron R. Voelker, Brent Komer, and Chris Eliasmith. Representing spatial relations with fractional binding. In *41st Annual Meeting of the Cognitive Science Society*, Montreal, QC, 2019. Cognitive Science Society.

- [60] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering, September 2018.
- [61] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large Language Models: A Survey, February 2024. arXiv:2402.06196.
- [62] Igor Nunes, Mike Heddes, Tony Givargis, Alexandru Nicolau, and Alex Veidenbaum. GraphHD: efficient graph classification using hyperdimensional computing. In *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe*, DATE '22, pages 1485–1490, Leuven, BEL, May 2022. European Design and Automation Association.
- [63] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 2024. Publisher: IEEE.
- [64] T.A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641, May 1995.
- [65] Tony A. Plate. *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. CSLI Publications, USA, March 2003.
- [66] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), January 2020. Publisher: JMLR.org.
- [67] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.
- [68] Daniel Rasmussen and Chris Eliasmith. A spiking neural model applied to the study of human performance and cognitive decline on Raven’s Advanced Progressive Matrices. *Intelligence*, 42:53–82, 2014. Publisher: Elsevier Inc.
- [69] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, August 2021.

- [70] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense Reasoning about Social Interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China, November 2019. Association for Computational Linguistics.
- [71] Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Le Hou, Kevin Clark, Stephen Pfohl, Heather Cole-Lewis, Darlene Neal, Mike Schaeckermann, Amy Wang, Mohamed Amin, Sami Lachgar, Philip Mansfield, Sushant Prakash, Bradley Green, Ewa Dominowska, Blaise Aguerre y Arcas, Nenad Tomasev, Yun Liu, Renee Wong, Christopher Semturs, S. Sara Mahdavi, Joelle Barral, Dale Webster, Greg S. Corrado, Yossi Matias, Shekoofeh Azizi, Alan Karthikesalingam, and Vivek Nataraajan. Towards Expert-Level Medical Question Answering with Large Language Models, May 2023.
- [72] Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990. Publisher: Elsevier.
- [73] Sarvesh Soni and Kirk Roberts. Evaluation of Dataset Selection for Pre-Training and Fine-Tuning Transformer Language Models for Clinical Question Answering. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5532–5538, Marseille, France, May 2020. European Language Resources Association.
- [74] Robyn Speer, Joshua Chin, and Catherine Havasi. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), February 2017.
- [75] Aarohi Srivastava and et al. Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, January 2023.
- [76] Terrence C. Stewart, Yichuan Tang, and Chris Eliasmith. A Biologically Realistic Cleanup Memory: Autoassociation in Spiking Neurons. *Cognitive Systems Research*, 12:84–92, 2011.
- [77] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve

- Them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada, July 2023. Association for Computational Linguistics.
- [78] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.
- [79] Yi Tay, Jason Wei, Hyung Chung, Vinh Tran, David So, Siamak Shakeri, Xavier Garcia, Steven Zheng, Jinfeng Rao, Aakanksha Chowdhery, Denny Zhou, Donald Metzler, Slav Petrov, Neil Houlsby, Quoc Le, and Mostafa Dehghani. Transcending Scaling Laws with 0.1% Extra Compute. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1471–1486, Singapore, December 2023. Association for Computational Linguistics.
- [80] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models, July 2023. arXiv:2307.09288.
- [81] Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, January 2024. Publisher: Nature Publishing Group.

- [82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [83] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, February 2018.
- [84] Aaron R. Voelker, Peter Blouw, Xuan Choo, Nicole Sandra-Yaffa Dumont, Terrence C. Stewart, and Chris Eliasmith. Simulating and Predicting Dynamical Systems With Spatial Semantic Pointers. *Neural Computation*, 33(8):2033–2067, July 2021.
- [85] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. SuperGLUE: a stickier benchmark for general-purpose language understanding systems. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 3266–3280. Curran Associates Inc., Red Hook, NY, USA, December 2019.
- [86] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.
- [87] Kuan Wang, Yuyu Zhang, Diyi Yang, Le Song, and Tao Qin. GNN is a Counter? Revisiting GNN for Question Answering. In *International Conference on Learning Representations*, October 2022.
- [88] Zhen Wang. Modern Question Answering Datasets and Benchmarks: A Survey, June 2022. arXiv:2206.15030.
- [89] Adina Williams, Nikita Nangia, and Samuel Bowman. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.
- [90] Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. Reasoning or Reciting? Exploring

the Capabilities and Limitations of Language Models Through Counterfactual Tasks. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1819–1862, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

- [91] Yichong Xu, Chenguang Zhu, Shuohang Wang, Siqu Sun, Hao Cheng, Xiaodong Liu, Jianfeng Gao, Pengcheng He, Michael Zeng, and Xuedong Huang. Human Parity on CommonsenseQA: Augmenting Self-Attention with External Attention. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, volume 3, pages 2762–2768, July 2022.
- [92] Yichong Xu, Chenguang Zhu, Ruochen Xu, Yang Liu, Michael Zeng, and Xuedong Huang. Fusing Context Into Knowledge Graph for Commonsense Question Answering. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1201–1207, Online, August 2021. Association for Computational Linguistics.
- [93] Michihiro Yasunaga, Antoine Bosselut, Hongyu Ren, Xikun Zhang, Christopher D. Manning, Percy S. Liang, and Jure Leskovec. Deep Bidirectional Language-Knowledge Graph Pretraining. *Advances in Neural Information Processing Systems*, 35:37309–37323, December 2022.
- [94] Michihiro Yasunaga, Jure Leskovec, and Percy Liang. LinkBERT: Pretraining Language Models with Document Links. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8003–8016, Dublin, Ireland, May 2022. Association for Computational Linguistics.
- [95] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546, Online, June 2021. Association for Computational Linguistics.
- [96] Zi Ye, Yogan Jaya Kumar, Goh Ong Sing, Fengyan Song, and Junsong Wang. A Comprehensive Survey of Graph Neural Networks for Knowledge Graphs. *IEEE Access*, 10:75729–75741, 2022.
- [97] Zhangdie Yuan, Songbo Hu, Ivan Vulić, Anna Korhonen, and Zaiqiao Meng. Can Pretrained Language Models (Yet) Reason Deductively? In *Proceedings of the 17th*

Conference of the European Chapter of the Association for Computational Linguistics, pages 1447–1462, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics.

- [98] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics.
- [99] Xikun Zhang, Antoine Bosselut, Michihiro Yasunaga, Hongyu Ren, Percy Liang, Christopher D. Manning, and Jure Leskovec. GreaseLM: Graph REASoning Enhanced Language Models. In *International Conference on Learning Representations*, October 2021.
- [100] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A Survey of Large Language Models, October 2024. arXiv:2303.18223.
- [101] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.