Learning Both Expert and Universal Knowledge Using Transformers

by

Yuan Li

Department of Electrical and Computer Engineering
Duke University

Date: _____

Approved:

_____
Lawrence Carin, Supervisor

_____
Yiran Chen

_____
Hai Li

_____
Rong Ge

_____
Zhenqiang Gong

Dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the Graduate School of
Duke University

2022

ABSTRACT

Learning Both Expert and Universal Knowledge Using Transformers

by

Yuan Li

Department of Electrical and Computer Engineering
Duke University

Date: _____

Approved:

_____
Lawrence Carin, Supervisor

_____
Yiran Chen

_____
Hai Li

_____
Rong Ge

_____
Zhenqiang Gong

An abstract of a dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the Graduate School of
Duke University

2022

Copyright © 2022 by Yuan Li
All rights reserved

Abstract

The Transformer [1] has demonstrated superior performance in various natural language processing (NLP) tasks, including machine translation [2], language understanding [3], and text generation [4]. The proposed multihead attention mechanism provides strong flexibility for fusing contextual information and therefore facilitates long-range relation modeling. Further, Transformers have proved effective for learning universal knowledge at scale, and representative models are BERT [5], GPT [6, 7, 8], and their subsequent variants [9, 10, 11, 12, 13]. It is observed that the Transformer is more tolerant to the convergence plateau and is capable of scaling to more than one hundred billion parameters.

Despite these advances, we believe that the Transformer can be further pushed toward the two extremes of knowledge learning: expert knowledge and universal knowledge. On the one hand, professional knowledge, such as medical knowledge accumulated by humans through a large amount of education and practice, plays a vital role in professional disciplines. However, due to the various forms of expert knowledge (*e.g.*, knowledge graphs, textual templates, and tables of statistics) and the need to develop different Transformer models to deal with different forms of knowledge, there is an urgent need for a unified framework to efficiently encode and decode different types of knowledge. On the other hand, learning universal knowledge requires substantial training data and a large model size to absorb the information from unlabeled data in a self-supervised manner. However, the existing self-supervised language models [5, 6, 7, 8] lack a structured encoding of the input and therefore fail to generate plausible text in a controllable way. Moreover, learning from high-dimensional input, such as image pixels, is challenging for the Transformer due to the heavy computational consumption and sparse semantic information represented by

the pixels [14].

In this proposal, we address these challenges by first defining a unified formulation for acquiring both expert and universal knowledge and then developing several novel Transformer models and their variants, including the Graph Transformers, the Variational Autoencoders (VAEs) implemented by the Transformer architecture, and the Visual-Linguistic Masked Autoencoders (VL-MAEs) for learning visual representation with additional language supervision. Finally, the techniques developed within this proposal will alleviate the burden and lower the entry bar of learning with universal knowledge and expertise for ML researchers and practitioners. They will also reduce the cost of research.

Acknowledgements

I'd like to express my sincere gratitude to my advisor, Prof. Lawrence Carin, who has guided my research throughout my whole Ph.D. period. You have always been giving me enough freedom to pursue research topics I like while giving me the irreplaceable support when I need help.

I also want to thank my dissertation exam committee members: Prof. Hai Li, Prof. Yiran Chen, Prof. Rong Ge, and Prof. Neil Gong. Your comments are very valuable to my research.

I would like to acknowledge the valuable discussions, suggestions, guidance, and collaboration from my lab mates and friends at Duke.

Last but not least, thank my family for your unconditional love and support!

Contents

Abstract **iv**

Acknowledgements **vi**

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**

1.1 Overview 1

1.2 Literature Review 3

1.2.1 Learning with Expertise 3

1.2.2 Pre-trained Language Models Using Transformers 4

1.2.3 Transformers in the Vision Domain 5

1.3 Contribution of this Dissertation 6

1.4 Dissertation Outline 8

I Learning Expert Knowledge **10**

2 Graph Transformers **11**

2.1 Introduction 11

2.2 Related Work 14

2.2.1 Medical Report Generation 14

2.2.2 Graph Neural Network 14

2.2.3 Hybrid Retrieval-Generation Approach 14

2.3 Graph Transformers (GTRs) 15

2.3.1 Inter-Graph Message Passing 16

2.3.2 Intra-Graph Message Passing 18
2.3.3 GTR as A Module 18
2.3.4 GTR for Multiple Domains 19
2.4 Knowledge-Driven Encode, Retrieve, Paraphrase 20
2.4.1 Encode: Visual Feature to Knowledge Graph 21
2.4.2 Retrieve: Knowledge Graph to Template Sequence 22
2.4.3 Paraphrase: Template Sequence to Report 22
2.4.4 Disease Classification 23
2.4.5 Learning 24
2.5 Experiments and Results 24
2.5.1 Dataset 24
2.5.2 Abnormality Definition and Collection 25
2.5.3 Template Definition and Collection 26
2.5.4 Visual Features Extraction 27
2.5.5 Evaluation Metrics 27
2.5.6 Training Details 28
2.5.7 Baselines for Abnormality and Disease Classification 28
2.5.8 Baselines for Medical Report Generation 29
2.5.9 Results and Analyses 30
2.6 Conclusion 34
3 Learning to Synchronize **35**
3.1 Introduction 35
3.2 Related Work 37
3.2.1 ML for Systems 37

3.2.2 Synchronization System Autotuning 38
3.2.3 Automatic ML Parallelization 38
3.3 Problem Definition 39
3.3.1 Background 39
3.3.2 Problem Formulation 39
3.3.3 Search Space 40
3.4 Learning to Synchronize 41
3.4.1 Domain Adaptive Simulator 43
3.4.2 Knowledge-Guided Search 48
3.4.3 Low-Cost Optimization using Transfer Learning 51
3.5 Experiments and Results 52
3.5.1 Models 53
3.5.2 Clusters 53
3.5.3 Baselines 54
3.5.4 Public Datasets 54
3.5.5 Simulator Training Dataset and Hyperparameters 56
3.5.6 End-to-End Results and Ablation Studies 56
3.5.7 Transferring Trained Simulators 60
3.6 Conclusion 63

II Learning Universal Knowledge 64

4 OPTIMUS 65

4.1 Introduction 65
4.2 Related Work 68

4.2.1 Language Models (LMs) 68
4.2.2 Linguistic VAEs 68
4.3 Preliminaries 69
4.4 Label-Conditional Text Generation 71
4.5 Experiments and Results 72
4.5.1 Baselines 72
4.5.2 Evaluation 72
4.5.3 Dataset 73
4.5.4 Results and Analysis 73
4.6 Conclusion 74
5 Complementary Auxiliary Classifiers 75
5.1 Introduction 75
5.2 Related Work 77
5.2.1 Difference from Attribute Transfer 77
5.2.2 Differences from Prior Work 78
5.3 Preliminaries 78
5.3.1 Autoencoder for Conditional Text Generation 79
5.3.2 Adversarially Regularized Autoencoders 80
5.4 Complementary Auxiliary Classifiers 82
5.4.1 Pitfalls of ARAEs 82
5.4.2 Disentanglement versus Controllable Generation 83
5.4.3 Proposed Method: CARA 84
5.4.4 Generation Quality versus Conditional Accuracy 88
5.5 Experiments and Results 90

5.5.1 Datasets 90
5.5.2 Evaluation 92
5.5.3 Baselines 92
5.5.4 Training Details 93
5.5.5 Results and Analysis 94
5.6 Conclusion 97
6 Self-Supervised Commonsense Learning 98
6.1 Introduction 98
6.2 Related Work 101
6.2.1 Commonsense QA 101
6.2.2 Pre-trained Language Models (LMs) 103
6.2.3 Learning with Auxiliary Tasks 103
6.3 Problem Formulation 104
6.4 Proposed Method 105
6.4.1 Contrastive Relation Learning (CRL) 105
6.4.2 Jigsaw Puzzles (JPs) 108
6.4.3 Additional Tasks 109
6.4.4 Overall Objective 111
6.5 Experiments and Results 111
6.5.1 Datasets 111
6.5.2 Baselines 112
6.5.3 Implementation Details 113
6.5.4 Results and Analysis 114
6.6 Conclusion 120

7 Visual-Linguistic Masked Autoencoders **122**

7.1 Introduction 122

7.2 Related Work 125

7.2.1 Masked Language and Image Modeling 125

7.2.2 Joint Visual-Linguistic Modeling 126

7.2.3 Autoencoders 127

7.3 Approach 128

7.3.1 Preliminaries 128

7.3.2 Masked Visual-Linguistic Modeling (MVLM) 129

7.3.3 Visual-Linguistic Masked Autoencoders (VL-MAEs) 130

7.4 Experiments and Results 132

7.4.1 Prompt Engineering 133

7.4.2 Zero-Shot Classification 133

7.4.3 Ablation Study on Text Masking Ratio 136

7.5 Conclusion 137

8 Conclusion and Discussion **139**

Bibliography **140**

Biography **158**

List of Figures

2.1 A medical image report example. 12

2.2 Architecture of the Graph Transformer. 18

2.3 Architecture of KERP. 19

2.4 Visualization of the KERP result for the IU X-Ray dataset. 34

3.1 Learning to synchronize framework and RNN and GAT simulators. . 42

3.2 Comparison between AutoSync variants. 59

3.3 Transferring the trained simulators. 61

4.1 Illustration of OPTIMUS architecture. 70

4.2 Label-conditional text generation on Yelp dataset. 73

5.1 Possible solutions to the disentangled feature learning objective. . . . 83

5.2 Illustration of the proposed CARA method. 85

5.3 Schedule comparison. 95

6.1 An example of commonsense multiple-choice QA and elBERto pipeline. 99

6.2 SSL tasks. 106

6.3 Results on WIQA Out-of-Paragraph questions. 118

6.4 Prediction examples of elBERto, individual SSL tasks, and baseline. . 119

6.5 Falling cases on the WIQA and CosmosQA datasets. 120

7.1 Proposed model architecture. 129

7.2 Predicting image categories in a zero-shot manner using prompts. . . 133

7.3 Zero-shot word clouds. 135

7.4 Ablation study on the number of visible tokens. 136

List of Tables

2.1 Statistics of the CX-CHR and IU X-Ray datasets. 26

2.2 Classification AUC. 29

2.3 Automatic and human evaluation. 30

3.1 Models experimented with. 52

3.2 Cluster specifications experimented with. 55

3.3 Comparisons of different model instantiations of the simulator. 56

3.4 The per-iteration time statistics of 200 strategies found by RS and GA on (VGG16, A). 57

3.5 Studies on feature importance. 60

3.6 The target domain test accuracy under different transfer learning settings. 62

4.1 Label-conditional text generation on Yelp dataset. 72

5.1 Dataset statistics 90

5.2 Personality captioning results. 91

5.3 Style captioning results. 91

5.4 Hyperparameters settings of CARA and its variants on different datasets. 93

5.5 Topic-based question generation results on Yahoo dataset. 94

5.6 Sentiment transfer results on Yelp dataset. 96

5.7 Sentiment transfer results on Yelp dataset. 97

6.1 Dataset statistics. 112

6.2 Test accuracy on WIQA data. 114

6.3 Accuracy on CosmosQA data. 115

6.4 Accuracy on the ReClor dataset. 116

6.5 Comparison of individual self-supervised tasks, elBERto and baseline. 117

7.1 Pre-training dataset summary statistics. 133

7.2 Cosine similarity. 134

Chapter 1

Introduction

1.1 Overview

In recent years, the Transformer [1] has stood out by demonstrating superior performance in various natural language processing (NLP) tasks, such as machine translation [2], language understanding [3], and text generation [4]. The proposed multihead attention mechanism provides strong flexibility for fusing contextual information and thus facilitates long-range relation modeling. Further, Transformers have proved powerful for learning universal knowledge at scale. BERT [5] first proposes encoding language bidirectionally using a Transformer encoder and predicting masked tokens, while GPT [6, 7, 8] trains a Transformer decoder to predict the next words. To facilitate effective and efficient pre-training using the Transformer (in whole or in part) in this direction, algorithms and machine learning (ML) models, such as the multitask learner GPT-2 [7], the language model GPT-3 [8], the robustly optimized RoBERta [9], the sample-efficient Electra [12], the big transferable pre-trained BiT [15], and Vision Transformer (ViT)—the first pre-trained Transformer for visual representation [16]—have been created. These large-scale models have proved superior in acquiring universal human knowledge from the massive scale of unlabeled data and well-designed self-supervised learning (SSL) objectives, achieving new state-of-the-art performance on many downstream NLP and computer vision (CV) tasks, such as machine translation, language modeling, language understanding, question answering, and image classification [5, 10, 17, 18].

Despite these advances, we believe that it is necessary to promote research on the Transformer in several areas to further verify its intelligence. Specifically, we propose to decompose it into the learning of expert knowledge and universal knowledge.

Some structured expert knowledge, such as medical knowledge graphs that humans usually acquire through substantial education and practice, cannot be explicitly captured by general-purpose Transformers. Thus, to acquire such knowledge, a massive number of deep neural networks and learning approaches are developed to push ML models to ingest diverse types of knowledgeable experiences. In this proposal, we specifically focus on two representative domains: (1) the typical medical domain that contains substantial expert knowledge (*e.g.*, medical knowledge graphs, diagnosis patterns, and report writing templates) but is in short supply of publicly available training data and (2) the recently fast-growing AutoML domain where expert experience is essential to facilitate the automatic exploration of big searching spaces.

Conversely, learning universal knowledge is not just about increasing models and data sizes. We believe that three crucial aspects need to be carefully and heavily examined to facilitate the Transformer model's ability to act like a human. Specifically, they are (1) structured encoding and controlled generation, (2) commonsense reasoning and question answering, and (3) learning from high-dimensional input containing meaningless units, such as image pixels. First, the existing pre-trained Transformers lack explicit and smooth modeling of structures in a latent space, thereby inhibiting controlled natural language representation from an abstract level and generation based on a series of affecting factors, such as sentiments, topics, templates, arithmetic operations, and interpolation [19]. Second, increasing model and data sizes during pre-training does not necessarily improve ML models' commonsense reasoning ability, as no specific objectives are designed to improve in this aspect. Third, it is desired to enhance the Transformer to extract semantics from high-dimensional input that

consists of meaningless units, such as image pixels [14]. We are yet to see a pre-trained model of visual representation that exhibits prevalent domination across various downstream computer vision (CV) applications. The lack of semantics of image pixels may be a reason for the difficulty of SSL in the visual domain.

In this proposal, we aim to address the challenges mentioned above. First, we develop a unified formulation to account for learning diverse types of expert knowledge and acquiring universal knowledge via self-supervised learning. Within this formulation, we address the challenges respectively, including the Graph Transformer for imposing prior expert knowledge into model training, an automatic synchronization framework that facilities and scales data-parallel distributed training, a pre-trained VAE model for structured language modeling and generation, a complementary auxiliary classifier regularized autoencoder that alleviates the non-identifiability issue of previous approaches , an effective SSL framework containing two novel SSL objectives for commonsense learning, and finally, a visual-linguistic masked autoencoder that jointly learns visual and language representation.

1.2 Literature Review

1.2.1 Learning with Expertise

Multiple forms of expertise are widely utilized to enhance ML model training. For example, knowledge bases are leveraged for understanding and reasoning requiring external knowledge [20, 21]; knowledge constraints are utilized when human prior experience is essential for guiding problem solving [22]; structured templates [23, 24] are applied for report writing in expert domains; and predefined modeling [22] is also helpful for injecting human prior knowledge. Many application scenarios require

human expertise. For example, in the medical domain, expert knowledge is crucial for guiding medical practitioners to perform medical processes, such as utilizing templates to write admission notes and diagnosis reports [23, 24] and recommending medications according to the relationship between diseases and medication [25]. Experience is also a key factor when developing effective models for tasks or fields with little labeled data. For example, to find efficiently distributed training strategies [26], ML practitioners must reply on their prior experience or trial runs to explore a combination of various factors that affect training throughput. Existing methods cannot provide a comprehensive performance on various model types because their synchronization assumptions are too simple and they only provide a fixed form of synchronization strategy, for example, parameter server (PS) [27] for BytePS [28] and Allreduce for Horovod [29]. Additionally, different strategies often demonstrate dramatic performance differences when applied to different ML building blocks (shallow, deep, sparse, dense, *etc.*) [30, 31]

1.2.2 Pre-trained Language Models Using Transformers

Pre-training language models (LMs) aim to learn universal language representation through web-scale unlabeled data and pretext tasks. Recently, significant pre-trained models [5, 6, 7, 8, 11, 32, 33] have proved to be effective for downstream NLP tasks. There are two general categories of pre-trained LMs: autoencoding and autoregressive. Autoencoding models focus on the encoding process. Within this category, BERT [5] proposes masked language modeling and next-sentence prediction objectives. RoBERTa [9] examines the impact of many key hyperparameters and the training data size of BERT and subsequently improves its robustness. Albert [10] is a lightweight BERT that applies factorized embedding parameterization and cross-layer parameter sharing techniques to reduce memory consumption and speed up the training

of BERT. In the autoregressive category, GPT [6] first proposes using the generative objective to pre-train the Transformer. The subsequent GPT-2 [7] and GPT-3 [33] increase the model and data size, obtaining improved performance when transferring to various downstream NLP tasks. XLNet [11] proposes learning bidirectional contexts by maximizing the expected likelihood over all permutations of the factorization order. To combine language understanding and generation tasks in one unified framework, several model variants have been proposed, including UniLM [34], BART [13], and T5 [35]. Although significant performance improvement has been reported on a wide range of NLP tasks, these models lack explicit modeling of structures in a latent space, making it difficult to control natural language generation/representation from an abstract level.

1.2.3 Transformers in the Vision Domain

Following the success of BERT [5], GPTs [6, 7, 8] and other pre-trained models in the NLP domain [9, 10, 11, 14], how to adapt the Transformer to the visual input has obtained substantial interest in the research community. The ViT [16] and DETR [36] are first proposed to bridge convolutional neural network (CNNs), which were the dominant model in the vision domain over the last decade [14] and the Transformers. These two models leverage CNNs and circumvent the problem of using the Transformer to directly process high-dimensional input. However, ViT requires much more pre-training time to achieve performance on par with CNNs. The Swin Transformer [37] is later proposed to use a hierarchical structure, and shifted windows to compute visual representation. DINO [38] adopts a self-distillation architecture and learns class-specific features and unsupervised object segmentation. Masked Autoencoders [14] propose using an autoencoder and masked patch technique with a high masking ratio to effectively learn visual representation. It is observed that the

decoder in the autoencoder is significant for the learning of visual features. However, the study of instance-level visual representation learning is insufficient.

1.3 Contribution of this Dissertation

We summarize the contribution of this dissertation as follows:

1. We proposed Graph Transformers to effectively inject expert knowledge as a prior knowledge graph, to represent different modalities, such as images, language, and knowledge graphs, as graphs, and to allow effective transformation between any two modalities, such as from an image to a knowledge graph and from a knowledge graph to a sequence of words.

2. We developed an automatic and composable synchronization framework for data-parallel distributed training by first unifying a variety of synchronization-affecting factors, such as communication architectures [27, 30, 29, 39], message encoding methods [40, 41], parameter partitioning, and merging schemes [31, 42], then establishing a distributed training system that consider and allow tuning of all affecting factors, and finally, proposing a simulator guided by human expertise for effective search of the design space spanned by the affecting factors.

3. We participated in developing OPTIMUS—a large-scale VAE employing a Transformer encoder-decoder architecture for language modeling and controllable text generation. OPTIMUS adopts a latent space to organize sentences in a structured and ordered way, allowing manipulation of latent representations and generation conditioned on labels, such as sentiments and topics.

4. We present a complementary auxiliary classifier regularized auto-encoder, which has are two classifiers to predict labels: a classifier in the latent space is used

for disentangling feature learning, as in ARAE, and a complementary classifier in the observation space is proposed to encourage the generator to contain the conditional label. We implement the encoder and generator as Transformers. It solves the non-identifiability issue of ARAE and enhances label-conditional text generation and style transfer.

5. We made the first attempt to present five SSL tasks to empower pre-trained language models with a strong capacity to understand rich logics and knowledge for commonsense question answering (QA). Specifically, a novel contrastive relation learning task is proposed to help QA solvers to fully understand local influences between events, and a JP task is designed to encourage a better understanding of logical chains and hidden commonsense in long contexts. The proposed model surpasses all existing works in the three most challenging commonsense QA benchmarks, even those with explicit graph reasoning and knowledge retrieval.

6. We explored visual-linguistic masked autoencoders, which consists of a unified Transformer encoder-decoder architecture for visual-linguistic modeling. We then employ the proposed Masked Visual-Linguistic Modeling (MVLM) pretext task to integrate masked image and language modeling. When transferred to a zero-shot image classification task, our model generates accurate class labels for unseen images without any fine-tuning, demonstrating its effectiveness in aligning visual and linguistic information via masked visual-linguistic modeling.

1.4 Dissertation Outline

- In Chapter 2, we present the proposed Graphs Transformers for effective incorporation of human expertise and transformation between different modalities.

- In Chapter 3, we study the task of learning to synchronize for data-parallel distributed training. We introduce a unified design space spanned by the synchronization-affecting factors, a Transformer-based simulator to speed up estimation of throughput of synchronization strategies, and finally, an automatic algorithm involving expert knowledge constraints to productively search the design space.

- In Chapter 4, we show the work in a pre-training large-scale VAE model and transferring it to the downstream label-conditional text generation task.

- In Chapter 5, we demonstrate the proposed complementary auxiliary classifiers for label-conditional text generation which alleviates the non-identifiability issue of previous methods and improves controllable generation and style transfer.

- In Chapter 6, we introduce the proposed self-supervised commonsense learning framework for question answering, which consists of two novel SSL objectives, namely a contrastive relation learning task that requires a QA solver to identify semantically valid contexts and a Jigsaw Puzzle task that encourages a better understanding of logical chains in long contexts.

- in Chapter 7, we present our work on visual-linguistic masked autoencoders which leverage Transformer encoder-decoder architectures for scalable visual representation learning. Language supervision is further incorporated to enhance the model's capability in zero-shot image classification task.

- in Chapter 8, we conclude our dissertation and discuss potential future work.

Part I

Learning Expert Knowledge

Chapter 2

Graph Transformers

2.1 Introduction

Beyond the traditional image captioning task [43, 44, 45] that produces single-sentence descriptions, generating long and semantic-coherent stories or reports to describe visual contents (*e.g.*, images and videos) has recently attracted increased research interest [46, 47, 48] and is positioned as a more challenging and realistic goal toward bridging visual patterns with human linguistic descriptions. A particularly challenging aspect of modeling long narratives from visual content is striking a balance between knowledge discovery and language modeling [44]. Current visual text generation approaches tend to generate plausible sentences that look natural by the language model but are poor at finding visual groundings. Although some approaches have been proposed to alleviate this problem [49, 50, 46], most ignore the internal knowledge structure of the task at hand. However, most real-world data and problems exhibit complex and dynamic structures, such as intrinsic relations among discrete entities under nature's law [51, 52, 53]. A knowledge graph, as one of the most powerful representations of dynamic graph-structured knowledge [54, 55], complements learning-based approaches by explicitly modeling the domain-specific knowledge structure and relational inductive bias. A knowledge graph also allows for incorporating priors, which have been proven useful for tasks where universal knowledge is desired or certain constraints must be met [56, 57, 58, 59].

As an emerging practical task for long text generation, medical image report

Medical Image Report

Indication: 65-year-old male.

Finding: The cardiomediastinal silhouette is normal in size and contour. Masslike opacification of right apex. No pneumothorax or large pleural effusion. The osseous structures are grossly normal.

Impression: Worsening masslike opacification of right apex-suggesting worsening malignancy or malignancy with postobstructive pneumonia.

Figure 2.1: A medical image report example.

generation [23, 60] must satisfy more critical protocols and ensure the correctness of medical terminology usage. As shown in Figure 2.1, a medical report consists of a findings section describing detailed medical observations of both normal and abnormal features, an impression or conclusion sentence indicating the most prominent medical observation, and peripheral sections, such as a patient's information and indications. Among these sections, the findings section is considered the most important component and is expected to 1) cover the contents of key relevant aspects, such as heart size, lung opacity, and bone structure; 2) correctly detect any abnormalities and support the findings with details, such as the location and shape of the abnormality; and 3) describe potential diseases, such as effusion, pneumothorax, and consolidation.

When writing a medical image report, radiologists must first check a patient's images for abnormal findings and then write the report by following certain patterns and templates, adjusting statements in the templates for each individual case when necessary [61]. To mimic this procedure, we propose formulating medical report writing as a knowledge-driven encode, retrieve, and paraphrase (KERP) process. In particular, KERP first invokes an Encode module to transform the visual features of medical images into an abnormality graph, where each node represents a possible clinical abnormality designed by prior medical knowledge, and their features depict the semantics of the abnormality that can be observed from the visual input (*e.g.*,

normal or abnormal, size, and location). The correlation of abnormality nodes is further encoded as edge weights of the abnormality graph so that the relations among different abnormal findings are considered when making a clinical diagnostic decision. Then, KERP retrieves a sequence of templates according to the detected abnormalities via a Retrieve module. The words of the retrieved templates are further expanded and paraphrased into a report by a Paraphrase module, which enriches the templates with details and corrects any false information.

As most real-world data (*e.g.*, images, sequences of text tokens, knowledge graphs, and convolutional feature maps) can be represented as graphs, we further propose a novel and generic implementation unit—Graph Transformer (GTR), which dynamically transforms among multi-domain graph-structured data. We further equip GTR with an attention mechanism for learning a robust graph structure and for incorporating a prior knowledge structure. By invoking GTR, KERP can robustly transform visual features to an abnormality graph (with the Encode module), then to sequences of templates (with the Retrieve module), and lastly to sequences of words (with the Paraphrase module).

We conducted extensive experiments on two medical image report datasets [62]. Our KERP achieved the state-of-the-art performance on both datasets under both automatic evaluation metrics and human evaluations. KERP also achieved the best performance on abnormality classification. The experiments showed that KERP not only generates structured and robust reports supported with accurate abnormality prediction but also produces explainable attentive regions, which is crucial for interpretative diagnosis.

2.2 Related Work

2.2.1 Medical Report Generation

Machine learning for healthcare [63, 64, 65] has been widely recognized in both academia and industry as an area of high impact and potential. The automatic generation of medical image reports, as one of the key applications in the field, is gaining increasing research attention [64, 60]. This task differs from others, such as summarization (as summaries tend to be more diverse, without clear templates or internal knowledge structure) and image captioning (where a single sentence is usually desired).

2.2.2 Graph Neural Network

Graph neural networks (GNN) have gained increasing research attention [66, 67, 68]. However, most existing methods learn to encode the input feature into higher-level features through selective attention over the object itself [69, 70, 71], whereas our method works on multiple graphs and models not only the data structure within the same graph, but also the transformation rules among different graphs.

2.2.3 Hybrid Retrieval-Generation Approach

The combination of traditional retrieval-based and modern generation-based methods for generating long texts [72, 23] has attracted greater research interest. Similar methods include paraphrasing a sampled text to certain styles [73, 74, 75]. Among all of the similar work, our study differs from Retrieve, Rerank, Rewrite [72] in that 1) our method has an encoding procedure that explicitly learns the graph structure of medical abnormalities and 2) the retrieval and reranking are formulated as one

comprehensive process in our method and implemented via a novel and generic unit called Graph Transformer (GTR).

2.3 Graph Transformers (GTRs)

We start by describing GTRs, which transform a graph into another graph to encode features into higher-level semantics within the same graph type or to translate the features of one graph (*e.g.*, a knowledge graph) into another one (*e.g.*, a sequence of words). First, we represent a graph as $G = (V, E)$. Here, $V = \{\mathbf{v}_i\}_{i=1:N}$ is a set of nodes, with each $\mathbf{v}_i \in \mathbb{R}^d$ representing a node's feature of dimension d, and N being the number of nodes in the graph. $E = \{e_{i,j}\}_{i,j=[1,N]}$ is the set of edges between any possible pair of nodes. Here, we study the setting in which each edge is associated with a scalar value indicating the closeness between the nodes, while it is straightforward to extend this formalism to other cases in which edges are associated with non-scalar values, such as vectors.

The GTR takes a graph, $G = (V, E)$, as input, and outputs another graph, $G' = (V', E')$. Note that G and G' are two different graphs and can have different structures and characteristics (*e.g.*, $N \neq N'$, $d \neq d'$, and $e_{i,j} \neq e'_{i,j}$). This differs from many previous methods [66, 67, 71] which are restricted to the same graph structures. For both the source and target graph, the set of nodes V and V' have been given beforehand (*e.g.*, the vocabulary size if the considered graph consists of sequences, abnormality nodes if the considered graph is an abnormality graph). We consider two scenarios for the edges among graph nodes: 1) the edges are provided previously, and denoted as e_{s_i,t_j} where s_i is the i_{th} node of the source graph and t_j is the j_{th} node of the target graph; 2) the edges are not provided, and thus source and target nodes are represented as fully connected with uniform weights. We assume that e_{s_i,t_j} has been

normalized to avoid the notation of averaging.

There are two types of message passing in GTR: from the source graph to the target graph (inter-graph message passing), and within the same graph (intra-graph message passing).

2.3.1 Inter-Graph Message Passing

To learn the information contained in the source graph, the features of source nodes are transformed and passed to target nodes with their corresponding edge weights. The formulation can be written as:

$$\mathbf{v}'_j = \mathbf{v}'_j + \sigma(\sum\nolimits_{i=1}^{N} e_{s_i,t_j} \mathbf{W}_s \mathbf{v}_i), \tag{2.1}$$

where σ is a nonlinear activation and $\mathbf{W}_s$ is a projection matrix of size $d' \times d$.

Considering that the edge information between the source and target graphs may not be available in many cases (*e.g.*, translating a sequence of words into another sequence of words), we propose to learn edge weights automatically using an attention mechanism [3]. In this way, target-node updating is enabled to consider the varying importance of the source nodes. Specifically,

$$\hat{e}_{s_i,t_j} = \text{Attention}(\mathbf{W}_s^a \mathbf{v}_i, \mathbf{W}_t^a \mathbf{v}'_j), \tag{2.2}$$

where $\hat{e}_{s_i,t_j}$ is the attention weight of the edge from source node i to target node j; $\mathbf{W}_s^a$ and $\mathbf{W}_t^a$ are the weights in the attention mechanism to project the nodes features of the source graph and target graphs, respectively, into a common space of dimension q; and Attention: $\mathbb{R}^q \rightarrow \mathbb{R}$ is the attention mechanism that transforms the two projected features $\mathbf{W}_s^a \mathbf{v}_i, \mathbf{W}_t^a \mathbf{v}'_j \in R^q$ into a scalar $\hat{e}_{s_i,t_j}$ that serves as the

edge's attention weight. In our experiments, Attention is parameterized as a scaled dot-product operation with multi-head attention [3].

The attention weights are normalized over all source nodes for each target node and denote the relative importance of each source node to the target node. The formulation can be written as:

$$\hat{e}_{s_i,t_j} = softmax_{s_i}(\hat{e}_{s_i,t_j}) = \frac{\exp\left(\hat{e}_{s_i,t_j}\right)}{\sum_{k=1}^{N} \exp\left(\hat{e}_{s_k,t_j}\right)}. \tag{2.3}$$

Once obtained, the normalized attention coefficients are be combined with prior edge weights to pass the features of connected source nodes to target nodes. The combined features serve as the target node's updated features with source graph knowledge encoded. We adopt weighted sums of the prior and learned attention edge weights as the final edge weights. Other methods, such as multiplication of the learned and prior edge weights followed by softmax, also work; however, in our experiments, we observed that the first method offers better performance and avoids under-fitting. The formulation can be written as:

$$\tilde{e}_{s_i,t_j} = \lambda e_{s_i,t_j} + (1-\lambda)\hat{e}_{s_i,t_j}, \tag{2.4}$$

where λ is a user-defined weight controlling the importance of prior and learned edges. If λ is set to 1, the edges between the source and target graphs are fixed, and no attention mechanism is required. The formulation is then the same as Equation 2.1. If λ is set to 0, the edges between the source and target graphs are completely learned by the model. With the updated weight, one can obtain updated target node features via Equation 2.1.

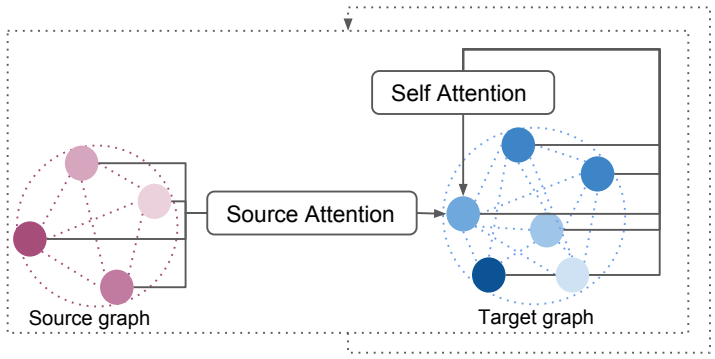


Figure 2.2: Architecture of the Graph Transformer. GTR evolves a target graph by recurrently performing Source Attention on a source graph and Source Attention on itself. The darkness of each graph node indicates the degree of attention the target node pays to it.

2.3.2 Intra-Graph Message Passing

Intra-graph message passing aims to model the correlation among nodes of the same graph and fuse features according to the closeness between them. Specifically, a target node is updated by combining its features with those of neighboring nodes. The formulation can be written as:

$$\mathbf{v}'_j = \mathbf{v}'_j + \sigma(\sum\nolimits_{i=1}^{N'} \tilde{e}_{i,j} \mathbf{W}_t \mathbf{v}'_i), \tag{2.5}$$

where $\mathbf{W}_t$ is the weight for projecting the features of the target nodes from dimension d to the output dimension. To learn the edge weights through the attention mechanism, one can directly apply Equations 2.1-2.4 by changing the notation of the source and target nodes to be on the same graph.

2.3.3 GTR as A Module

As shown in Figure 2.2, we formulate GTR as a module by first concatenating intra-graph message passing and inter-graph message passing into one step (that is, first conducting message passing within the target graph, and then from one or multiple

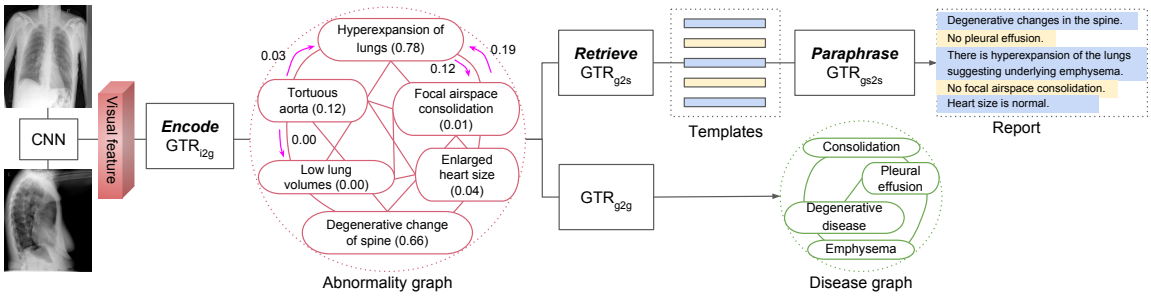


Figure 2.3: Architecture of KERP. Image features are first extracted from a CNN and further encoded as an abnormality graph via Encode GTR_{i2g}. Retrieve GTR_{g2s} decodes the abnormality graph as a template sequence, the words of which are then retrieved and paraphrased by Paraphrase GTR_{gs2s} as a generated report. Simultaneously, a GTR_{g2g} decodes the abnormality graph as a disease graph and predicts disease categories via extra classification layers. In the abnormality graph, the values within parentheses are probabilities of the corresponding nodes predicted by extra classification layers, taking the nodes' latent semantic features as input. Values along the directed arrows indicate the attention scores of source nodes on target nodes.

source graphs), and then stacking multiple such steps into one module to progressively convert target graph features into high-level semantics.

2.3.4 GTR for Multiple Domains

As most real-world data types (*e.g.*, images, sequences, graphs) can be formulated as graph-structured, it is easy to extend GTR for transformation among various data domains. In particular, we define GTR_{i2g} as a variant of GTR for transforming image features into graph's features; GTR_{g2g} as that for transforming from a graph to another graph; GTR_{g2s} as that for graph input and sequence output; and GTR_{gs2s} as that for graph and sequence input and sequence output. We use the variants of GTR building blocks for a knowledge-driven encode, retrieve, and paraphrase (KERP) method for medical report generation, which is described next.

2.4 Knowledge-Driven Encode, Retrieve, Paraphrase

To write a medical image report, radiologists first check a patient's images for abnormal findings, then follow certain patterns and templates and adjust statements within the templates for each individual case as necessary [61]. To mimic this procedure, we propose formulating medical report writing as a process of encoding, retrieval, and paraphrasing. In particular, we first compile an off-the-shelf abnormality graph that contains a large range of abnormal findings. We consider frequent abnormalities stemming from thoracic organs—such as "disappearance of the costophrenic angle", "low lung volumes", and "blunted costophrenic angle"—as nodes of the abnormality graph. These abnormalities compose a major part of medical reports, and their detection quality would greatly impact the accuracy of the generated reports. We also compile a template database that consists of a set of frequently occurring sentences that describe different abnormalities in the abnormality graph. For example, "there is hyperexpansion of the lungs and flattening of the diaphragm consistent with COPD" is a template for describing "hyperexpansion of the lungs", "flattening of the diaphragm," and "COPD."

Then, we design separate modules to encode visual features as an abnormality graph, retrieve templates based on the detected abnormalities, and rewrite templates according to case-specific scenarios. As described in Figure 2.3, a set of images are first fed into a CNN to extract visual features, which are then transformed into an abnormality graph via Encode GTR_{i2g}. Retrieving GTR_{g2s} decodes the abnormality graph as a template sequence, the words of which are then retrieved and paraphrased by Paraphrase GTR_{gs2s} as a generated report.

In addition, we design a disease graph containing common thoracic diseases (*e.g.*, nodules, pneumonia, and emphysema) which are commonly diagnosed based on single

or combined abnormalities. For example, atelectasis may be concluded if "interval development of bandlike opacity in the left lung base" is present; consolidation and atelectasis may exist if there are "streaky and patchy bibasilar opacities" and "triangular density projected over the heart" without "typical findings of pleural effusion or pulmonary edema." In parallel to generating reports in the proposed model architecture, GTR_{g2g} is employed to transform the abnormality graph into a disease graph to predict common thoracic diseases (right lower column of Figure 2.3), which can be useful as concluding information for medical reports.

2.4.1 Encode: Visual Feature to Knowledge Graph

The Encode module aims to encode visual features as an abnormality graph. Assume that an input image is encoded with a deep neural network into feature $\mathbf{X} \in R^{WH,d_X}$ where W, H, and d_X are the width, height, and feature dimension, respectively. An abnormality graph is represented as a set of nodes of size N with initialized features. The latent features of each node can be used to predict the occurrence of the abnormality via an additional classification layer. By applying the variant of GTR variant for image input and graph output, (GTR_{i2g}), the updated node features can be written as:

$$\mathbf{h}_u = \text{GTR}_{\text{i2g}}(\mathbf{X}), \tag{2.6}$$

$$\mathbf{u} = \text{sigmoid}(\mathbf{W}_u \mathbf{h}_u), \tag{2.7}$$

where GTR_{i2g} is the formulation of the variant of GTR for image input and graph output described on page 18, and $\mathbf{W}_u$ is a linear projection to transform a latent feature u into a 1-d probability. $\mathbf{h}_u = (\mathbf{h}_{u_1}; \mathbf{h}_{u_2}; ..., \mathbf{h}_{u_N}) \in R^{N,d}$ is the set of latent features of nodes, where d is the feature dimension. $\mathbf{u} = (u_1, u_2, ..., u_N), y_i \in \{0, 1\}, i \in \{1, ..., N\}$

denotes a binary label for abnormality nodes.

2.4.2 Retrieve: Knowledge Graph to Template Sequence

Once the knowledge graph of the abnormality attributes is obtained, it can be used to guide the retrieval process to harvest templates. A sequence of templates is represented as $\mathbf{t} = (t_1, t_2, ..., t_{N_s})$, where N_s is the maximum length of a template sequence (also the maximum number of sentences) and t_i is the index of templates, where $i \in \{1, ..., N_s\}$. By applying the variant of GTR for the graph input and sequential output, denoted GTR_{g2s}, the retrieved template sequence can be obtained as follows:

$$\mathbf{h}_t = \text{GTR}_{\text{g2s}}(\mathbf{h}_u), \tag{2.8}$$

$$\mathbf{t} = \arg\max \text{Softmax}(\mathbf{W}_t \mathbf{h}_t), \tag{2.9}$$

where GTR_{g2s} denotes the formulation of the variant of GTR for the graph input and sequence output, $\mathbf{W}_t$ is a linear projection to transform latent feature to template embeddings, and $\arg\max \text{Softmax}$ is the operation that selects the index with maximum value after performing the Softmax operation.

2.4.3 Paraphrase: Template Sequence to Report

The Paraphrase module serves two purposes: 1) refining templates with enriched details and possibly new case-specific findings; 2) converting templates into more natural and dynamic expressions. The first purpose is achieved by modifying information in the templates that is not accurate for specific cases, and the second is achieved by robust language modeling for the same content. These two goals supplement each other to generate accurate and robust reports.

Given the retrieved sequence of templates $\mathbf{t} = (t_1, t_2, ..., t_{N_s})$, the rewriting step generates a report consisting of a sequence of sentences $\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_{N_s})$ by subsequently attending to template words and the encoded knowledge graph described on page 4. Each sentence $\mathbf{r}_i = (w_{i1}, w_{i2}, ..., w_{iN_w}), i \in \{1, .., N_s\}$ consists of a sequence of N_w words.

$$\mathbf{h}_w = \mathrm{GTR}_{\mathrm{gs2s}}(\mathbf{h}_u, \mathbf{t}), \tag{2.10}$$

$$\mathbf{R} = \arg\max \mathrm{Softmax}(\mathbf{W}_w f(\mathbf{h}_w)), \tag{2.11}$$

where $\mathrm{GTR}_{\mathrm{gs2s}}$ denotes the formulation of the GTR variant for graph and sequence input and sequence output, f denotes the operation of reshaping $\mathbf{h}_w$ from $R^{N_s, N_w, d}$ to $R^{N_s * N_w, d}$, $\mathbf{W}_w$ is a linear projection to transform the latent feature into word embeddings, and $\arg\max \mathrm{Softmax}$ selects the index with maximum value after performing the Softmax operation.

2.4.4 Disease Classification

Abnormality attributes are frequently used to diagnose diseases. Let $\mathbf{z} = (z_1, z_2, ..., z_L)$ be a one-hot representation of disease labels, where L is the total number of disease labels and $z_i \in \{0, 1\}, i \in \{1, ..., L\}$. The multi-label disease classification can be written as:

$$\mathbf{h}_z = \mathrm{GTR}_{\mathrm{g2g}}(\mathbf{h}_u), \tag{2.12}$$

$$\mathbf{z} = \mathrm{sigmoid}(\mathbf{W}_z \mathbf{h}_z), \tag{2.13}$$

where $\mathrm{GTR}_{\mathrm{g2g}}$ denotes the formulation of the variant of GTR for graph input and graph output and $\mathbf{W}_z$ is the linear projection to transform the disease nodes feature

into a 1-d probability.

2.4.5 Learning

During paraphrasing, the retrieved templates $\mathbf{t}$, instead of latent feature $\mathbf{h}_t$, is used for rewriting. Sampling the templates of the maximum predicted probability breaks the connectivity of the differentiable back-propagation of the whole encode-retrieve-paraphrase pipeline. To alleviate this issue, we first train the Paraphrase module with ground truth templates, and then with sampled templates generated by the Retrieval module. This minimizes the negative effect of dis-connectivity and offers better test-time performance by letting the model accommodate its own generated templates. Moreover, given that the templates serve as starting points instead of ground truth for rewriting, the prediction of templates does not have to be highly accurate, as the Paraphrase module needs to learn to paraphrase any suitable templates.

2.5 Experiments and Results

2.5.1 Dataset

We conduct experiments on two medical image report datasets. Detailed statistics of IU X-Ray and CX-CHR dataset is shown in Table 2.1.

Indiana University Chest X-Ray Collection (IU X-Ray)

IU X-Ray [62] is a public dataset consisting of 7,470 chest x-ray images paired with their corresponding diagnostic reports. Each patient has two images (a frontal view and a lateral view) and a report that includes impression, findings, and indication sections. We preprocess the reports by tokenizing and converting the text into lower-

cases. We select only tokens with a minimum frequency of 3, resulting in 1,185 unique tokens that cover over 99.0% of word occurrences in the corpus. We collect 80 abnormalities and 87 templates for the IU X-Ray dataset.

CX-CHR

CX-CHR is a private dataset of chest X-ray images with corresponding Chinese reports collected from a professional medical institution for health checking. The dataset consists of 35,609 patients and 45,598 images. Each patient has one or multiple chest x-ray images taken from different points of view (*e.g.*, frontal and lateral), and a corresponding Chinese report. We select patients with no more than two images and obtain 33,236 patient samples in total, covering over 93% of the dataset. We preprocess reports by tokenizing using Jieba [76], and filtering tokens whose frequencies are no less than 3, resulting in 1,479 unique tokens. We collect 155 abnormalities and 362 templates for the CX-CHR dataset.

On both datasets, we randomly split the data by patients into training, validation and testing by a ratio of 7:1:2. There is no overlap between patients in different sets.

2.5.2 Abnormality Definition and Collection

The abnormal findings generally take these forms: 1) the presence of abnormal attributes of an object (*e.g.*, bibasilar consolidation), 2) the absence of typical attributes (*e.g.*, disappearance of the costophrenic angle), 3) an abnormal change of the object's shape (*e.g.*, enlarged heart size), or 4) an abnormal change in the object's location (*e.g.*, elevated left hemidiaphragm). We consider all clinical abnormalities stemming from thoracic organs as nodes in the abnormality graph. We use the normalized co-occurrence of abnormality attributes computed from the training corpus as prior

Table 2.1: Statistics of the CX-CHR and IU X-Ray datasets. "#" indicates "number of".

Statistics	IU X-Ray	CX-CHR
#Patients	3,867	35,609
#Images	7,470	45,598
#Diseases	14	14
#Abnormalities	80	155
#Templates	87	362
#Abnormal templates	66	290
#Normal templates	21	72
Vocabulary size	1185	1479
Max. #sentences	18	24
Max. sentence length	42	38
Max. sentence length (by tokens)	42	18
Max. report length	173	216
Avg. #sentences	4.637	9.120
Avg. sentence length	6.997	7.111
Avg. report length	32.450	64.858
Avg. predicted #sentences	4.420	9.984
Avg. predicted sentence length	4.751	7.309
Avg. predicted report length	21.003	66.045

edge weights for the knowledge graph, the normalized co-occurrence of disease labels as prior edge weights for the disease graph, and the normalized co-occurrence of any abnormality attribute and disease labels as edge weights from source knowledge graph nodes to target disease nodes. The abovementioned coefficients for prior edge weights are all set to 0.9.

2.5.3 Template Definition and Collection

We first collect sentences that describe each abnormality and have frequencies no less than a threshold in the training corpus. Then, we manually group sentences of the same meaning and select the most frequent one from each group as a template. To generate the template sequence and report the word sequence, we assume no prior edge weights for the target graph or prior edge weights from the source graph to the target graph. We assume uniform prior edge weights among all graph nodes and use

a weight of 1.0 on the learned edge weights.

2.5.4 Visual Features Extraction

To fairly compare with all baselines, we extract visual features for both datasets from the last convolutional layer of a DenseNet [77] jointly pre-trained on CX-CHR and the publicly available ChestX-ray8 dataset [63]. The IU X-Ray dataset is not used for pre-training due to its relatively small size; we add an additional lateral layer as in the Feature Pyramid Network [78] for the last three dense blocks and additional convolutional layers to transform the feature dimension to 256. We extract features from the last convolutional layer of the second dense block, yielding $64 \times 64 \times 256$ feature maps. These feature maps have a higher resolution and contain more location information without expanding the total feature size too much beyond that directly extracted from the last layer of DenesNet (*e.g.*, $16 \times 16 \times 1{,}024$ feature maps).

2.5.5 Evaluation Metrics

We use the area under the curve (AUC) to evaluate the performance of abnormality and disease classification. To evaluate medical report generation, we use two kinds of evaluation metrics: 1) automatic metrics, including CIDEr [79], ROUGE-L [80], and BLEU [81] and 2) human evaluation metrics, for which we randomly select 100 sampled testing results for each method and conduct surveys through Amazon Mechanical Turk. Each survey question yields a ground truth report and asks candidates to choose among reports generated by different models. The criteria are the degrees of correctness of abnormal findings, language fluency, and content coverage compared to the ground truth report. A default choice is provided in case of no or all reports are preferred. We collect results from five participants and compute the average

preference percentage for each model excluding default choices.

2.5.6 Training Details

We first train Encode for abnormality classification and then separately train Retrieve with fitted templates supervision and Paraphrase with fitted templates as input and ground truth reports as output for fixed Encode. Then, we fine-tune Paraphrase using sampled templates from Retrieve. As Retrieve may not predict the same length or order of sentences as the ground truth, we re-order the ground truth sentences by choosing the smallest edit distance to the corresponding retrieved template sentence. We use learning rates of $1e^{-3}$ for training and $1e^{-5}$ for fine-tuning and reduce these rates by a factor of 10 when encountering the validation performance plateau. We use early stopping, a batch size of 4, and a drop-out rate of 0.1 for all training.

GTR_{i2g} for Encode has three graph message-passing layers and six heads in multi-head attention. GTR_{g2s} and GTR_{gs2s} (for Retrieve and Paraphrase, respectively) have six graph message-passing layers and eight heads in multi-head attention. The dimension of all hidden features and embeddings is set to 256. The coefficients for prior edge weights, if provided, are all set to 0.9. The word embedding of Retrieve and Paraphrase, and the projection matrix $\mathbf{W}_w$ to project latent features onto the word distribution, are shared. We implement our model using PyTorch and train it on two GeForce GTX TITAN GPUs.

2.5.7 Baselines for Abnormality and Disease Classification

We compare the performance of abnormality and disease classification with a DenseNet [77] trained on classifying the same set of abnormality or disease labels. For abnormality classification, we compare our method by solely training on abnormality classification

Table 2.2: Classification AUC.

Dataset	Model	Abnormality	Disease
IU X-Ray	DenseNet	0.612	0.646
	Ours-1Graph	0.674	-
	Ours-2Graphs	**0.686**	**0.726**
CX-CHR	DenseNet	0.689	0.800
	Ours-1Graph	0.721	-
	Ours-2Graphs	**0.760**	**0.862**

(Ours-1Graph) and jointly training on both abnormality and disease classification (Ours-2Graphs). For disease classification, we compare with DenseNet.

2.5.8 Baselines for Medical Report Generation

On both datasets, we compare with four state-of-the-art image captioning models: CNN-RNN [82], LRCN [83], AdaAtt [84], Att2in [45]. We use the same visual features and train/test split on both datasets, respectively. Besides the abovementioned generation-based methods in which no prior knowledge is provided, we also compare with retrieval-based methods for which prior templates are available to further examine how well templates work for structured medical report generation and for different datasets. We create the HardRetrieval baseline, which generates a fixed report for all testing samples. The fixed report is selected to contain the most common heterogeneous sentences that describe normal cases. By so doing, HardRetrieval mimics a brute-force system that predicts common and "safe" sentences if no confident abnormality is observed. For the published results, we compare with CoAtt on the IU X-Ray dataset, which uses different visual features stated in [60]. We re-train CoAtt using the same split as other baselines and fix its fine-tuning on ResNet classification for a fair comparison with all other models by taking fixed visual features. We also compare with HRGR-Agent [23], which uses the CX-CHR dataset features stated in [23]. We

Table 2.3: Automatic and human evaluation for the IU X-Ray (upper part) and CX-CHR datasets (lower part), as compared with CNN-RNN [82], LRCN [83], AdaAtt [84], Att2in [45], CoAtt [60], and HRGR-Agent [23]. * indicates re-training and evaluation on our data split.

Dataset	Model Class	Model	CIDEr	ROUGE-L	BLEU-1	BLEU-2	BLEU-3	BLEU-4	Hit (%)
IU X-Ray	Generation	CNN-RNN	0.294	0.307	0.216	0.124	0.087	0.066	–
		LRCN	0.285	0.307	0.223	0.128	0.089	0.068	–
		AdaAtt	0.296	0.308	0.220	0.127	0.089	0.069	–
		Att2in	0.297	0.307	0.224	0.129	0.089	0.068	–
		CoAtt*	0.277	**0.369**	**0.455**	0.288	0.205	**0.154**	
	R+G	HRGR-Agent	**0.343**	0.322	0.438	**0.298**	**0.208**	0.151	
		KER	0.318	0.335	0.455	0.304	0.210	–	
		KERP	0.280	0.339	0.482	0.325	0.226	0.162	–
CX-CHR	Retrieval	HardRetrieval	1.820	0.584	0.641	0.544	0.470	0.413	–
	Generation	CNN-RNN	1.580	0.577	0.590	0.506	0.450	0.411	–
		LRCN	1.588	0.577	0.593	0.508	0.452	0.413	–
		AdaAtt	1.568	0.575	0.588	0.503	0.446	0.409	–
		Att2in	1.566	0.576	0.587	0.503	0.446	0.408	25.937
	R+G	HRG	2.800	0.588	0.630	0.547	0.497	0.463	–
		HRGR-Agent	**3.530**	0.618	0.668	0.585	0.530	**0.486**	–
		KER	0.817	0.552	0.609	0.489	0.400	0.335	–
		KERP	2.850	**0.618**	**0.673**	**0.588**	**0.532**	0.473	**67.820**

retrain HRGR-Agent with our visual features on IU X-Ray and our train/test split. To study the effectiveness of individual modules of KERP, we compare KERP with its variant KER, which excludes the Paraphrase module and only uses retrieved templates as prediction results.

2.5.9 Results and Analyses

Abnormality and Disease Classification

The AUCs of abnormality and disease classification are shown in Table 2.2. Ours-2Graphs outperforms baselines for both datasets on abnormality and disease classification. DenseNet performs worst on both datasets, being outperformed by Ours-2Graphs by more than 6%. This demonstrates that, by incorporating prior medical knowledge and learning internal medical knowledge structure, our method can distill useful features to correctly classify abnormalities and diseases. Given the high performance of Ours-2Graphs, we conduct the following experiments using the knowledge graph trained with both abnormality and disease labels as initialization.

Medical Report Generation on The IU X-Ray Dataset

The results are shown in the upper part of Table 2.3. First, it is interesting to observe that when only comparing HardRetrieval and the first four generation-based baselines (namely CNN-RNN, LRCN, AdaAtt, and Att2in; rows 2–6 of Table 2.3), HardRetrieval outperforms all others in ROUGE-L and BLEU-n by large margins but achieves a 0 CIDEr score. Note that the CIDEr score considers the inverse document frequency (idf) of each word in the whole evaluated corpus, whereas ROUGE-L and BLEU-n match only n-grams within the evaluated and ground truth sentences for each testing sample. In particular, the shorter the considered n-gram, (*e.g.*, 1-gram in BLEU-1 and 2-gram in BLEU-2), the more easily the model boosts evaluation performance by generating popular and seemingly correct words. However, by considering idf, the CIDEr score can differentiate important and rare words from popular but insignificant ones. Due to the unbalanced distribution of normal and abnormal findings in medical reports, which is demonstrated extremely clearly in the IU X-Ray dataset due to its relatively small size, CIDEr appears to be a better evaluation metric. Our experimental results show that, by only generating normal and popular sentences, HardRetrieval can boost ROUGE-L and BLEU-n performances on the IU X-Ray dataset. However, when considering the CIDEr score, the effectiveness of generation-based models stands out by being able to predict significant words for abnormal cases.

Another generation-based baseline model, CoAtt (7_{th} row of Table 2.3), which uses another set of visual features, achieves the highest BLEU-1,2,3 scores but a relatively low CIDEr score, indicating that the model may be primarily repeating popular sentences without differentiating abnormal findings for rare cases. On the other hand, KERP achieves the best CIDEr score by outperforming all other models by a margin of more than 0.278. This demonstrates that KERP can identify abnormal findings more

accurately and generate correct reports instead of ones that are seemingly natural but lack visual groundings. This also verifies the effectiveness of the proposed abnormality graph for explicitly encoding observed abnormal findings from visual features. Finally, KER outperforms all baselines and has only slightly lower results than KERP. This verifies the functionality of the Retrieve module for accurately retrieving templates. KERP also outperforms KER slightly on CIDEr and ROUGE-L but relatively large on BLEU-n, which shows that the Paraphrase module can effectively transform templates into more common language descriptions without changing the accurate information contained in them.

Medical Report Generation on The CX-CHR Dataset

First, as shown in the lower part of Table 2.3, HardRetrieval outperforms all generation-based models according to all evaluation metrics and serves as a strong baseline. It also indicates that the CX-CHR dataset is relatively less unbalanced than the IU X-Ray dataset since, by generating popular and normal sentences, HardRetrieval can boost not only its ROUGE-L and BLEU-n scores, but also its CIDEr score. Second, KERP outperforms other generation-based models by large margins, demonstrating the effectiveness of combining retrieval and generation mechanisms for generating medical reports. Third, KERP outperforms HRGR-Agent in ROUGE-L and BLEU-1,2,3; although KERP achieves a lower CIDEr score than HRGR-Agent, this is reasonable, since HRGR-Agent is fine-tuned by Reinforcement Learning using CIDEr as a reward. Moreover, KERP achieves much better performance on abnormality prediction than [23]) (Table 3), demonstrating a significant contribution to clinical diagnosis. Finally, it is interesting to observe that KER performs worse than generation-based methods in most metrics; this may due to the templates for CX-CHR being designed for concision and to cover a large range of different abnormal findings, rather than being

natural and common. Thus, the use of only retrieved templates does not lead to high performance; however, the overall high performance of KERP confirms that Paraphrase module can distill accurate information in the retrieved templates and paraphrase them into more common and natural descriptions. It also shows that learning radiologists conventional and general writing style is as important as learning correct information for medical report generation.

Human Evaluation

We conduct human evaluation as a supplement to automatic evaluation, the result of which is shown in the last column of Table 2.3. KERP outperforms the compared baseline for both datasets, demonstrating its ability to generate fluent and accurate reports.

Qualitative Analysis

The visualization result of KERP on the IU X-Ray dataset is shown in Figure 2.4. The generated reports align significantly with ground truth reports and correspond with the visualized attention maps. For example, the generated report for the first sample correctly describes "right greater than left" airspace disease, and the attention map of the frontal view shows a red region over the right upper lung greater than that over the left lower lung (the redness indicates the degree of attention). The result demonstrates that KERP is capable of generating accurate reports as well as explainable attentive regions.

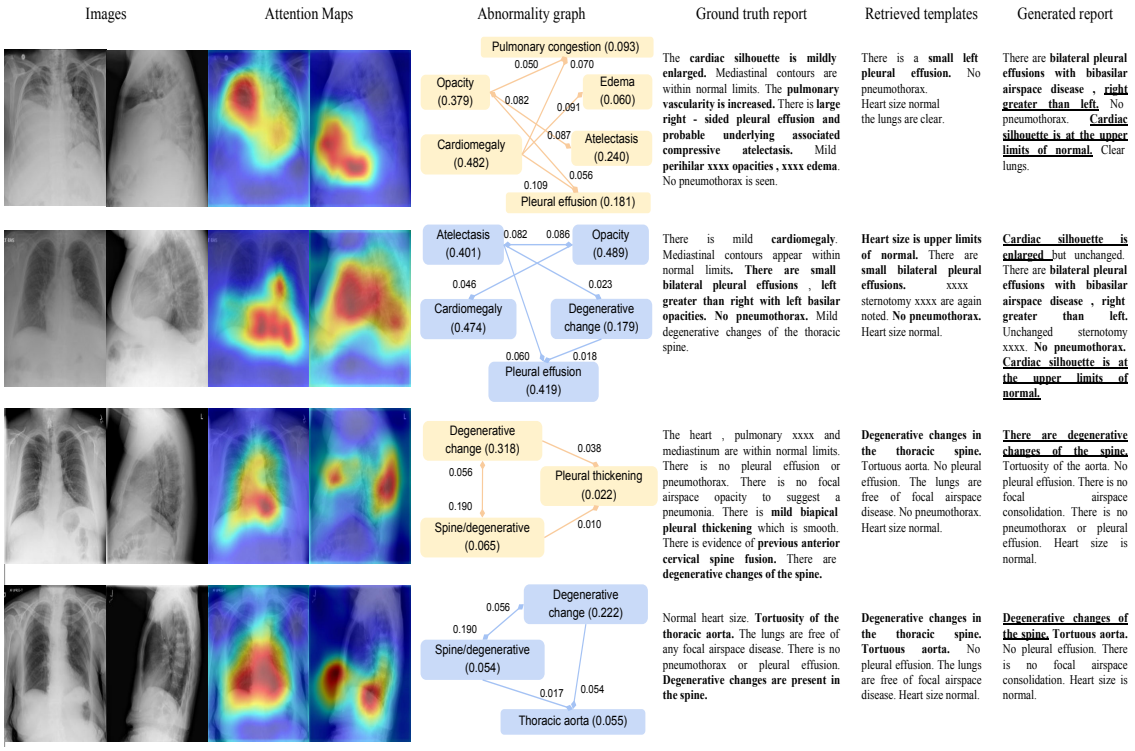


Figure 2.4: Visualization of the KERP result for the IU X-Ray dataset. Bold text indicates alignment between the generated text and ground truth reports; underlined text indicates correspondence of the generated text (specifically the location description) with the visualized attention maps of either the first/frontal or second/lateral views. In the abnormality graph, the values inside parentheses are the predicted probabilities of corresponding nodes. We select edges whose attention scores are greater than or equal to 0.05, or are the highest from neighboring nodes to each node, and visualize the attention scores along the directed arrows.

2.6 Conclusion

We proposed a novel method for accurate and robust medical report generation, called the Knowledge-driven Encode, Retrieve, and Paraphrase (KERP) method. A generic implementation unit called Graph Transformer (GTR)—the first attempt to transform multi-domain graph-structured data via an attention mechanism—was also proposed. KERP achieved state-of-the-art performance on two medical image report datasets and generated accurate attribute predictions, dynamic medical knowledge graphs, and explainable location references.

Chapter 3

Learning to Synchronize

3.1 Introduction

Larger models trained on larger datasets have recently been shown to be beneficial and influential in deep learning (DL). To address the associated computational complexity, data-parallel training [28, 39, 30] was introduced and demonstrated remarkable success in scaling up model training to thousands of GPUs [85], while taking into account billions of parameters [5, 7]. Data parallelism divides and distributes large datasets to multiple worker devices, calculates gradients for each worker on its independent data split, and synchronizes gradients of all workers at the end of each iteration.

To facilitate synchronization in data-parallel training, a variety of strategies and system implementations have been presented, including systems designed for specific communication architectures [27, 30, 29, 39], message encoding methods [40, 41], parameter partitioning, and merging schemes [31, 42]. However, in order to achieve the desired data-parallel speedup, the synchronization strategy must be compatible with the statistical and algorithmic properties of the model as well as the cluster specification. Bipartite parameter servers, for example, are appropriate for models whose sparsity structure generates "hot spots" [86, 28, 31], whereas collective all-reduce outperforms other communication architectures when the majority of distributed communication occurs between GPUs [29, 85]. Existing systems struggle to provide excellent all-round performance on diverse models due to their oversimplified assumptions about the synchronization and rigid application of fix-formed synchro-

nization strategies (*e.g.,* parameter server (PS) [28, 87] for BytePS [27], Allreduce for Horovod [29]), which ignore the characteristics of models or clusters. More importantly, when applied to different ML building blocks (shallow, deep, sparse, dense, *etc.*), different strategies frequently exhibit sharp performance differences [30, 31]. As a result, the burden of choosing the best strategy for the model of interest falls on ML practitioners, who may lack domain expertise on the trade-offs between these systems. Given the combinatorial number of options for various synchronization factors, (*e.g.,* architecture, variable partitioning, and placement configuration), manually searching for the optimal strategy is prohibitively expensive, and the search must be redone every time a new model is developed.

The goal of this chapter is to address these issues by asking whether it is possible to automate the selection of the optimal synchronization strategy, given a model and cluster specification? To that end, we identify a number of synchronization-affecting factors in data-parallel distributed DL. We create a valid and large strategy space spanned by multiple factors by factorizing the strategy with respect to each trainable building block of a DL model. We build an end-to-end pipeline, AutoSync to help us navigate the space and find the optimal strategy. AutoSync reduces the search space by leveraging domain knowledge about synchronization systems and is equipped with a domain adaptive simulator that combines principled communication modeling and data-driven ML models to estimate the runtime of strategy proposals without launching real distributed execution. We investigate the transferability of trained simulators across diverse models and resource configurations, in order to cut practical development costs even further.

We test AutoSync on a wide range of models and clusters, demonstrating that there are numerous strategies in the proposed space that outperform hand-optimized systems by a large margin. On several, difficult-to-parallelize model architectures (*e.g.*,

NCF [88], BERT [5] and VGG16 [89]), AutoSync can effectively develop solutions that cut the training time by 1.2–1.6x than hand-optimized ones while staying under an acceptable budget. AutoSync simulators can be trained on cheaper trial data acquired on smaller models or clusters, and then utilized to derive strategies without the need for extra training for larger models or costlier clusters, thanks to transfer learning. We also collect a dataset of roughly 10,000 data points containing (model, resource, strategy) tuples and their accompanying runtime on real clusters as a bonus contribution. We make the dataset available to the public in order to stimulate further research. [1]

3.2 Related Work

3.2.1 ML for Systems

There has been a surge in interest in using ML to solve system problems. Mirhoseini *et al.* [90, 91] create reinforcement learning (RL) frameworks to identify where nodes in dataflow graphs should be placed. Paliwal *et al.* [92] use a combination of RL and genetic algorithms (GA) to reduce the cost of NN graphs execution for compilers. AutoTVM [93] builds an ML-based pipeline that generates operator implementations that are superior to those designed by hand. AutoSync is part of this line of work as it uses ML to optimize data-parallel synchronization and subsequently addresses the unique challenges that come with it.

[1] The data and code are available at `https://github.com/petuum/autodist`.

3.2.2 Synchronization System Autotuning

Many data-parallel ML systems are capable of autotuning to some extent. Horovod [29] and ByteScheduler [27] both use Bayesian optimization (BO) [94] but introduce adjustable knobs and credit size, respectively, to autotune their values. Parallax [31] uses trial data to train a 3-parameter linear model to find the best partitioning for sparse variables in PS. This work focuses on autotuning one or two hyperparameters of a specific strategy. However, AutoSync contrasts them by co-optimizing a generic and holistic representation of synchronization. Among them, the closest to ours is AutoTVM [22]. We draw on AutoTVM for inspiration, but we address a fundamentally different problem – data-parallel auto-distribution of ML training on clusters – which necessitates the problem-specific design of strategy representations, features, runtime simulations, and so on.

3.2.3 Automatic ML Parallelization

The ultimate goal of distributed ML is automate the parallelization of ML programs. FlexFlow [42, 95] provides the SOAP representation to define NN layer partitioning methods, and an MCMC-based algorithm to find optimum partitioning configurations. TOFU [96] is concerned with the division of finer-grained computing processes in DL dataflow graphs. They intersect with Autosync because data parallelism (which AutoSync concentrates on) may be described equivalently as partitioning all the layers/operations along their batch dimension. However, in addition to partitioning, AutoSync models' numerous other synchronization-affecting elements, such as communication architectures, sharding, and merging schemes, differentiate us from others in this line of work.

3.3 Problem Definition

3.3.1 Background

A DL model is represented by its dataflow graph $\mathcal{G} = \{(V_{\mathcal{G},\theta}, V_{\mathcal{G},o}), E_{\mathcal{G}}\}$ where $V_{\mathcal{G}}$ are nodes in $\mathcal{G}$ that include trainable variables $V_{\mathcal{G},\theta} = \{v_i\}_{i=1}^{|V_{\mathcal{G},\theta}|}$ or computational operations $V_{\mathcal{G},o}$, and $E_{\mathcal{G}}$ are tensors (edges) transferred between nodes. For simplicity, we use V, which is equivalent to $V_{\mathcal{G},\theta}$, to denote the set of variables. Further, we define a cluster as a device graph $\mathcal{D} = \{V_{\mathcal{D}}, E_{\mathcal{D}}\}$, where $V_{\mathcal{D}} = \{d_p\}_{p=1}^{|V_{\mathcal{D}}|}$ represents devices (*e.g.*, CPUs or GPUs), and $E_{\mathcal{D}} = \{b_{i,j}\}$ is a symmetric matrix with the entry $b_{i,j}$, reflecting the connectivity (*e.g.*, bandwidth) between d_i and d_j. In data-parallel training, we replicate $\mathcal{G}$ on all devices and update each trainable variable v_i by aggregating the stochastic gradients $\nabla_{v_i}(\mathcal{G}, X_p)$ computed by each worker device d_p on its data partition X_p, following $v_i^{(t+1)} \leftarrow v_i^{(t)} + \epsilon \sum_{p=1}^{P} \nabla_{v_i^{(t)}}(\mathcal{G}, X_p)$, for $v_i \in V_{\mathcal{G},\theta}$. Since devices are dispersed throughout the cluster, achieving the aggregate necessitates synchronization support, which gathers updates $\nabla_{v_i^{(t)}}$ and provides all devices the shared access to a consistent version of $v_i^{(t+1)}$. Existing systems seek to optimize some single factor to speed up the synchronization, neglecting the fact that the optimal of each factor varies greatly with $\mathcal{G}$ and $\mathcal{D}$. Without domain experiences, it is difficult for ML practitioners to identify appropriate synchronization strategies for their models of interest.

3.3.2 Problem Formulation

Alternatively, we pose the strategy selection as an optimization problem, in which the per-iteration runtime (*e.g.*, time taken to process a batch on all nodes of the entire cluster, equivalent to system throughput), is denoted as f and is minimized, given $\mathcal{G}$

and $\mathcal{D}$, by solving the following:

$$\min_{\mathcal{S}} f(\mathcal{G}, \mathcal{D}, \mathcal{S}), \text{ s.t. } \mathcal{C}, \tag{3.1}$$

where $\mathcal{S}$ denotes a model and resource dependent representation of the synchronization strategy and $\mathcal{C}$ is a set of constraints (developed in §3.4.2). Analyzing this problem requires continuous characterizations of $\mathcal{S}$ and f, which are not available. In light of the recent AutoML advances [97, 98, 22], we define a domain-specific space that takes into account several synchronization-affecting factors, and we use search-based methods to discover the near-optimal $\mathcal{S}^*$.

3.3.3 Search Space

The following must be considered when building the search space. To begin, rather than optimizing a single factor in a piecemeal fashion, as is customary in present systems, we seek a unified space spanning many synchronization-affecting factors, in order to capture the intricacies between them and their dynamics with various $\mathcal{G}$ and $\mathcal{D}$ via co-optimization. On the other hand, we wish to build direct correspondences between $\mathcal{S}$ and each variable of $\mathcal{G}$ in synchronization so that the strategy may adapt with specific variable-wise mathematical properties.

As a result, we begin by decomposing existing fixed-formed systems/strategies into the following orthogonal factors. [2] (1) Variable partitioning, represented as $\boldsymbol{p}_i = [p_i^j]_{j=1}^{k_i}$ for the variable v_i, where k_i denotes the number of tensor dimensions of v_i, and p_i^j represents the partition degree on the jth axis. (2) Variable placement, defined as $\{d_i\}_{i=1}^m \subset V_{\mathcal{D}}$, which is the set of devices in which the variable resides, and because

[2] Since we focus on synchronous training and optimizing system throughput, we exclude optimizations beyond data-parallel training (*e.g.,* operation partitioning in model-parallel training), or introduce deviation of parameter updates (*e.g.,* staleness).

the placement is a single device, v_i is shared across all devices. (3) Synchronization architecture: we define two types of architecture primitives, parameter server (PS) and collective communication (CC), as well as their architecture-specific semantics for each. In PS, reduction hierarchy is used to indicate whether parameters are conveyed hierarchically (*e.g.,* from a central CPU to multiple GPUs co-located on the same machine). In CC, we define a merge group as where communication primitives assigned to the same group are merged and communicated via a single message, and device order, which specifies the message passing order between devices (*e.g.*, tree and ring). (4) Finally, message encoding and decoding, denoted as c_i for v_i, introduces compression or decompression schemes to represent how messages are processed prior to (after) communication, enabling optimizations [99, 30, 40] that exploit structures (*e.g.,* low-rank) exhibited in the messages to reduce message size and speed up network transfer.

Based on these factors, the strategy is described as $\mathcal{S} = \{s_i\}_{i=1}^{|V|}$ where s_i, as a sub-strategy, includes the discrete choices of above factors for each $v_i \in \mathcal{G}$. Because we determine whether or not to partition variables before any other aspects, a sub-strategy must be generated for each variable partition. The many factors span a combinatorial space that expands in size as $\mathcal{G}$ and $\mathcal{D}$ grow. The learning to synchronize framework is then developed to approximate the optimal $\mathcal{S}^*$.

3.4 Learning to Synchronize

Despite the huge space, solving Eq. 3.1 presents a new challenge: searching for $\mathcal{S}^*$ necessitates assessing f for each strategy proposal, which needs distributed execution on clusters and is prohibitively costly. To make the search more tractable, we offer the learning to synchronize framework (Figure 3.1), which includes two new components:

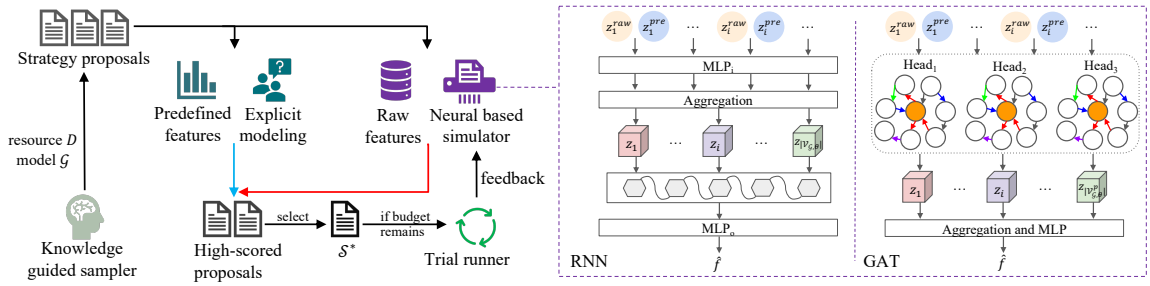


Figure 3.1: Left: Learning to synchronize framework. Initially, the simulator utilizes domain-agnostic features (Section 3.4.1) to explicitly estimate the runtime so to select promising strategies for evaluation (the blue line). After trials, the real runtime data are feedbacked to train the ML-based simulator to adapt to specific $\mathcal{G}, \mathcal{D}$, enhancing its capability in differentiating high-quality strategies. Gradually, the ML-based simulator takes over and directs the search (the red line). **Right**: Illustrations of the RNN and GAT simulators.

runtime simulation of arbitrary $\mathcal{S}, \mathcal{G}, \mathcal{D}$ (§3.4.1) and knowledge-guided strategy search (§3.4.2).

In particular, we create a domain-adaptive simulator to predict f in order to reduce real execution. Estimation is made achievable (without training data) by first constructing features that are independent of $\mathcal{G}, \mathcal{D}$. The features, which use specified modeling to represent crucial runtime aspects may be generalized to any unseen $\mathcal{G}$ and $\mathcal{D}$. Then, we improve them by employing ML models and numerous raw features derived from specific $\mathcal{G}, \mathcal{D}$. We use prior knowledge about synchronization systems to instantiate constraints $\mathcal{C}$ in Eq. 3.1 to further direct the search to the subspace where suitable strategies may be located. The final framework, AutoSync, handles $\mathcal{S}^*$ in another manner. The simulator leverages the training-free features in the initial phase to suggest the first batch of strategy candidates, which are then executed on real clusters to acquire ground truth runtime. The low-shot data is sent back into the simulator to train it to adapt to $\mathcal{G}, \mathcal{D}$, resulting in better proficiency in discriminating high-performing strategies. The process is repeated until the optimization budget for real distributed cluster evaluation has been depleted. Next, we will detail the designs of the simulator and guided search.

3.4.1 Domain Adaptive Simulator

The simulator accepts $(\mathcal{G}, \mathcal{D}, \mathcal{S})$ as input and calculates the runtime for each iteration (equivalent to throughput). Because variable partitioning will alter $V_{\mathcal{G},\theta}$, we first infer the new set of variables $V'_{\mathcal{G},\theta}$ based on $\mathcal{G}, \mathcal{S}$, and let the simulator work with each variable $v_i \in V'_{\mathcal{G},\theta}$, which contains variable shards after partitioning original variables. We define a particular $(\mathcal{G}, \mathcal{D})$ target as a domain. When equipped with domain-agnostic features, the simulator is capable of estimating the runtime of any $(\mathcal{G}, \mathcal{D})$ without training. This is accomplished by systematic runtime modeling in distributed execution.

Predefined Modeling

We use two contributing components to model the per-iteration runtime T of parallelizing $\mathcal{G}$ on $\mathcal{D}$: computation time T_{comp}, and parameter synchronization time T_{sync}. Since one component usually dominates the other in data-parallel training [100], we simply obtain T by $T = \max(T_{\text{comp}}, T_{\text{sync}})$. The reasoning is as follows: (1) since many runtime systems (such as TensorFlow [101] or PyTorch [102]) introduce scheduling or parallelization between communication and computation, in actuality, there are significant overlaps between the two components; (2) in data-parallel training, it is commonly observed that one component usually dominates the other [100]. As a result, considering the maximum of them as an estimation of total time is fair. T_{comp} and T_{sync} are factored with respect to variables in the same way that $\mathcal{S}$ is, and we estimate $T_{\text{comp}}(v_i), T_{\text{sync}}(v_i)$ for each v_i based on its s_i in $\mathcal{S}$. By profiling its equivalent operation on a single device, $T_{\text{comp}}(v_i)$ can be estimated. To compute $T_{\text{sync}}(v_i)$, we divide $V'_{\mathcal{G},\theta}$ into variables with PS as V^{PS} and collective communication as V^{CC}, and derive two analytic forms of $T_{sync}(v_i)$.

For $v_i \in V^{PS}$, if we denote the size of the message by v_i as $c_i(m_i)$ where the encoding/decoding scheme $c_i \in s_i$ has been applied on the original size m_i, and w_i as the number of worker nodes involved in synchronizing v_i, the communication time T^{PS}_{server} on the server hosting v_i, indexed as j, is as follows:

$$T^{PS}_{\text{server}}(v_i) = \underbrace{\sum_{k=1}^{w_i} \mathbb{I}_d(j,k) \cdot \frac{c_i(m_i)}{b_{j,k}} \cdot r_{i,k}^{\mathbb{I}_p}}_{\text{network transfer}} + \underbrace{\sum_{k=1}^{w_i} \mathbb{I}_d(j,k) \cdot r_{i,k}^{\mathbb{I}_p} \cdot \phi}_{\text{network overhead}} + \underbrace{\delta}_{\text{GPU kernel latency}}, \quad (3.2)$$

where $r_{i,k}$ is the number of replicas (*e.g.*, number of GPUs) of $\mathcal{G}$ on worker k, $\mathbb{I}_d(j,k)$ and $\mathbb{I}_p$ are true when server j and worker k locate on different machines and when hierarchical reduction is used, respectively. Moreover, ϕ and δ are network overhead and GPU kernel latency constants, respectively. Of the three, the first corresponds to sending messages from each worker k to the server j (and vice versa). The second then captures network overheads that linearly scale with the number of workers or replicas when $\mathbb{I}_p$ is false. Constant GPU memory latency is captured by the third term.

In addition to the formula, we can construct domain-agnostic features of v_i as $\boldsymbol{z}_i^{PS} = [\text{network transfer}, \text{coefficient of } \phi, \text{coefficient of } \delta]$. A communication bottleneck may be caused by the slowest transmission or the total time of transmissions, as synchronizing any v_i between any pair of nodes can happen simultaneously and is upper bounded by the multi-flow bandwidth. Thus, we define global features for estimating T^{PS}_{sync} as $\boldsymbol{z}^{PS} = \max\{\boldsymbol{z}_i^{PS} \text{ for } v_i \in V^{PS}\} \oplus \sum\{\boldsymbol{z}_i^{PS} \text{ for } v_i \in V^{PS}\}/|V^{PS}|$, where $\oplus$ is vector concatenation, and $\max, \sum$ are element-wise.

Similarly, for $v_i \in V^{CC}$, we consider 5 commonly used collective primitives: `AllReduce`, `ReduceScatter`, `AllGather`, `Broadcast`, and `Reduce` [103]. For example, when there are w workers and the device order in the substrategy s_i is a ring [29],

each primitive sends and receives $\frac{2(w-1)}{w}$, $\frac{w-1}{w}$, $\frac{w-1}{w}$, $1, 1$ times, respectively, in its applicable scenario for parameter synchronization (e.g. `AllReduce` for dense gradients or `AllGather` for sparse gradients [29]) . Therefore, we can obtain $T_{sync}^{CC}(v_i)$ as following:

$$T_{sync}^{CC}(v_i) = \underbrace{\mathbb{I}_1 \frac{2(w_i-1)c_i(m_i)}{w_i b_m} + \mathbb{I}_2 \frac{(w_i-1)c_i(m_i)}{w_i b_m} + \mathbb{I}_3 \frac{c_i(m_i)}{b_m}}_{\text{network transfer}} + w_i \cdot \phi + \delta, \tag{3.3}$$

where $b_m = \min_{(k_1,k_2)\in \text{ring}} b_{k_1,k_2}$ denotes the lowest bandwidth between devices in the ring because the throughput of a ring is restricted by the lowest bandwidth in the network [104]. $\mathbb{I}_1, \mathbb{I}_2, \mathbb{I}_3$ are true when `AllReduce`, `ReduceScatter` and `AllGather`, `Broadcast` and `Reduce` are activated, respectively. The formula is derived by counting how many times each message (i.e. gradients) needs to be passed through the ring while taking into account both network transfer overhead and the device-host memory swap latency. The total synchronization time for variables assigned with collective communication is $T_{sync}^{CC}(V^{CC}) = \sum_{i=1}^{|V^{CC}|} T_{sync}^{CC}(v_i)$.

The domain-agnostic features for $v_i \in V^{CC}$ are $\boldsymbol{z}_i^{CC} = [\text{network transfer, coefficient } of \phi, \text{coefficient of } \delta]$ from Eq. 3.3. The global features of T_{sync}^{CC} are $\boldsymbol{z}^{CC} = \max\{\boldsymbol{z}_i^{CC}$ for $v_i \in V^{CC}\} \oplus \sum\{\boldsymbol{z}_i^{CC}$ for $v_i \in V^{CC}\}/|V^{CC}|$.

Concatenating $\boldsymbol{z}^{PS}$ and $\boldsymbol{z}^{CC}$ results in the set of domain-agnostic features $\boldsymbol{z}^{pre}$ for $(\mathcal{G}, \mathcal{D}, \mathcal{S})$. We can either use the estimated T to rank different $\mathcal{S}$, or use the constructed features as inputs to ML models, which we elaborate next.

Domain-Specific Modeling

Upon acquiring trial data, we augment $\boldsymbol{z}^{pre}$ using raw features $\boldsymbol{z}^{raw}$ extracted from $\mathcal{G}, \mathcal{D}, \mathcal{S}$, and train ML models to capture their domain-specific characteristics. For each v_i, $\boldsymbol{z}_i^{raw}$ vectorizes attributes including variable placement, synchronization

architecture, encoding/decoding type, and merge group from $\mathcal{S}$, bandwidth and the number of replica devices of each node from $\mathcal{D}$, and variable size, dimensions, the sparsity of gradients, data types, and information of partitioned shards from $\mathcal{G}$. Combining $\boldsymbol{z}^{pre}$ and variable-specific raw features $\boldsymbol{z}_i^{raw}$, we adopt three different ML models: (1) a linear model, (2) a recurrent neural network (RNN), and (3) a graph attention networks (GAT) [105], to learn from trial data and make more accurate predictions of f.

Linear Model. Simply written as $\hat{f} = \boldsymbol{z}^{pre}\cdot$, the model introduces trainable weights and predicts the runtime $\hat{f}$ using only global features $\boldsymbol{z}^{pre}$.

RNN. We use an RNN to model different $\mathcal{G}$ with varying numbers of variables to inject $\boldsymbol{z}_i^{raw}$. First, the RNN concatenates $\boldsymbol{z}_i^{pre}$ with $\boldsymbol{z}_i^{raw}$. Then, it transforms the combined features via an MLP. The results are then combined into $|V_{\mathcal{G},\theta}|$ features corresponding to the variables in the $V_{\mathcal{G},\theta}$. A bidirectional LSTM scans them following the forward-backward propagation order preserved in $\mathcal{G}$. The formulation of leveraging a RNN with both predefined and raw features for runtime prediction is as follows:

$$\boldsymbol{z}_i = \mathrm{MLP}_i\left(\boldsymbol{z}_i^{raw} \oplus \boldsymbol{z}_i^{pre}\right), \tag{3.4}$$

$$\boldsymbol{h}_{i+1} = \mathrm{LSTM}\left(\boldsymbol{h}_i, \boldsymbol{z}_i\right), \tag{3.5}$$

$$\boldsymbol{h}'_{i+1} = \mathrm{LSTM}\left(\boldsymbol{h}'_i, \boldsymbol{z}_{|V'_{\mathcal{G},\theta}|+1-i}\right), \tag{3.6}$$

$$\hat{f} = \mathrm{MLP}_o([\boldsymbol{h}_{|V'_{\mathcal{G},\theta}|}, \boldsymbol{h}'_{|V'_{\mathcal{G},\theta}|}]), \tag{3.7}$$

where MLP_i projects the input features into a hidden representation space, and MLP_o projects the encoded features after LSTM fusion to the output logits. The LSTM walks through each $v_i \in V'_{\mathcal{G},\theta}$ strictly following their original forward (backward) order

in the computational graph to inject this information into the modeling. The RNN works on the transformed graph $\{V'_{\mathcal{G},\theta}, E'_{\mathcal{G}}\}$, which has additional partitioned variables compared to the original graph $\{V_{\mathcal{G},\theta}, E_{\mathcal{G}}\}$.

GAT. We use GAT to model the raw graph structure of $\mathcal{G}$. To do this, we prune $\mathcal{G}$ into a graph $\mathcal{G}^p = (V^p_{\mathcal{G},\theta}, E^p_{\mathcal{G}})$, with only variable (including partitioned) nodes and edges connecting them. Similar to RNN, each node's raw and predefined features are concatenated, and the features of all nodes and accompanying edge information are fed to a GAT for encoding graph structure as illustrated in Figure3.1 and as $\{\boldsymbol{z}_i\} = \text{GAT}(\{\boldsymbol{z}_i^{raw} \oplus \boldsymbol{z}_i^{pre} \text{ for } v_i \in V^p_{\mathcal{G},\theta}\}, E^p_{\mathcal{G}})$. The node features are then aggregated into graph-level ones, which are subsequently processed by an MLP to obtain the runtime estimation $\hat{f}$. The formulation of using a GAT with both predefined and raw features for runtime estimation is as follows:

$$\{\boldsymbol{z}_i\}_{i=1}^{|V'_{\mathcal{G},\theta}|} = \text{GAT}(\{\boldsymbol{z}_i^{raw} \oplus \boldsymbol{z}_i^{pre}, \text{ for } v_i \in V'_{\mathcal{G},\theta}\}, E'_{\mathcal{G}}), \tag{3.8}$$

$$\boldsymbol{z} = \frac{1}{|V'_{\mathcal{G},\theta}|} \sum (\{\boldsymbol{z}_i\}_{i=1}^{V'_{\mathcal{G},\theta}}), \tag{3.9}$$

$$\hat{f} = q(\boldsymbol{z}), \tag{3.10}$$

where GAT denotes the graph attention operations [106] applied on the graph $\{V'_{\mathcal{G},\theta}, E'_{\mathcal{G}}\}$ with $\boldsymbol{z}_i^{raw} \oplus \boldsymbol{z}_i^{pre}$ as node features. q is a linear layer that projects hidden representation to runtime estimation.

Training Objective

It can be challenging to accurately predict runtime because of uncertain factors on a distributed cluster. As such, we train the simulators with the pair-wise logistic

ranking loss [107, 22]:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbb{I}(f_i+\sigma\mathbb{I}_1(P_i,P_j)>f_j)\log(1+\exp(\hat{f}_j-\hat{f}_i)), \tag{3.11}$$

where f_i, f_j are ground truth runtimes, $\mathbb{I}$ is an indicator function, and n denotes the number of training examples. We add a penalty term $\sigma\mathbb{I}_1(P_i, P_j)$ to the original ranking loss, where σ is a non-negative threshold, P_i is the total number of partitions in $\mathcal{S}_i$, and $\mathbb{I}_1(x, y)$ outputs 1 when $x > y$, -1 when $x < y$, and 0 otherwise. This term additionally takes into account the intrinsic partitioning overheads and mitigates the tendency of heavily partitioning variables.

3.4.2 Knowledge-Guided Search

To evaluate a strategy, the search uses the simulator and proposes candidates with low predicted runtime $\hat{f}$. Then, a small number of qualified candidates are selected from a large set of proposals for trial execution, as shown in Figure 3.1. Two search algorithm variants are implemented here: random search and genetic algorithm (GA) [108].

Knowledge Constraints

Gird searching the whole strategy space is inefficient due to its exponential size. To limit the search inside promising regions, we implement the constraints $\mathcal{C}$ in Eq. 3.1 using two system design principles. (1) Load balancing constraint c_{lb}: This is vital to removing the communication bottleneck. When determining the placement for each $v_i \in V^{PS}$, c_{lb} enforces sampling from a multinomial distribution over all participating nodes in $\mathcal{D}$. Here, each node's probability of being chosen correlates to its current communication load and maximum bandwidth. So, nodes with higher available bandwidth are more likely to be sampled. This difference allows randomized solutions

to be generated while approximately maintaining a balanced status across all nodes. (2) Adjacent merging constraint c_{am}: The fusion of collection operations ought to correspond to the model forward-backward propagation order as in $\mathcal{G}$. This stands as merging two operations in the head and tail of the model prevents low-level scheduling overlapping communication and computation. The c_{am} is introduced to ensure it is more likely that variables adjacent to each other in $\mathcal{G}$ are grouped together.

Algorithm 1: Implementation of the load balance constraint c_{lb}

Inputs: $\mathcal{S}$, $\mathcal{D}$, $V'_{\mathcal{G},\theta}$

Function `load_balance_constraint(`$\mathcal{S}$, $\mathcal{D}$, $V'_{\mathcal{G},\theta}$`):`

- Set loads of each device in $\mathcal{D}$ as $l_{cur} = [0, ..., 0]$
- Sum up the bandwidth of devices in $\mathcal{D}$: $b_{all} = \sum_{i=1}^{|\mathcal{D}|} b_{i,i}$
- Sort $V'_{\mathcal{G},\theta}$ by byte size in descending order and get $V^{s'}_{\mathcal{G},\theta}$
- **for** v *in* $V^{s'}_{\mathcal{G},\theta}$ **do**
  - $l_{all} = \sum_{i=1}^{|\mathcal{D}|} l_{cur}[i]$
  - $logits = l_{all} * [b_{1,1}/b_{all}, \ldots, b_{|V_{\mathcal{D}}|,|V_{\mathcal{D}}|}/b_{all}] - l_{cur}$
  - $j \sim \text{Categorical}(\text{softmax}(logits))$
  - Set the placement of v in $\mathcal{S}$ as $d_j \in \mathcal{D}$
  - $l_{cur}[j] = l_{cur}[j] + \text{byte_size}(v)$

The two knowledge constraints, load balancing constraint c_{lb} and adjacent merging constraint c_{am}, are implemented together with the strategy sampler as regulations. In particular, the strategy sampler first generates a partial strategy $\mathcal{S}$, leaving two fields, *variable placement* (PS-based) and the *group* (collective-based) undecided. A global load balancer (c_{lb}) and group assigner (c_{am}) assign their values using randomized and approximate solutions, illustrated in Algorithm 1 and Algorithm 2, respectively. Note both constraints work on the set of variables $V'_{\mathcal{G},\theta}$ with partitioned shards, instead of the original set of variables $V_{\mathcal{G},\theta}$.

The two constraints help filter strategies that are obviously inefficient, but approximately maintain global load balance on the cluster and adjacent merging structures

Algorithm 2: Implementation of the adjacent merge constraint c_{am}

Inputs: $\mathcal{D}$, $V'_{\mathcal{G},\theta}$, $\mathcal{G}$, the number of total collective merging groups N

Function `adjacent_merge_constraint(`$\mathcal{D}$,$V'_{\mathcal{G},\theta}$,$\mathcal{S}$,N`):`

  Sort $V'_{\mathcal{G},\theta}$ based on their location in the backpropagation order indicated by $\mathcal{G}$, from input to loss function, and get $V^{s'}_{\mathcal{G},\theta}$

  Set loads $l_{cur} = [0, ..., 0]_{i=1}^{N}$ corresponding to N merge groups

  $l_{avg} = \frac{1}{N} \sum_{v \in V^{s'}_{\mathcal{G},\theta}} \text{byte_size}(v)$

  $g = 0$

  for v *in* $V^{s'}_{\mathcal{G},\theta}$ **do**

    if $l_{cur}[g] \geq l_{avg}$ **then**

      $g = g + 1$

    if $l_{cur}[g] < l_{avg}$ **then**

      Set the group of v in $\mathcal{S}$ as g

      $l_{cur}[g] = l_{cur}[g] + \text{byte_size}(v)$

correlated with the forward-backward propagation order. We show the c_{lb} and c_{am} empirically improve search efficiency and quality in §3.5.6.

Select Evaluation Candidates

We choose the final set of candidate in the same way as AutoTVM [22] by minimizing the weighted sum of $\hat{f}$ and internal similarity. The similarity can be assessed by comparing strategy representations or simulator concealed outputs. Formally, we solve the following optimization problem:

$$\min_{\{\mathcal{S}_i\}_{i=1}^{K} \subset \{\mathcal{S}_i\}_{i=1}^{M}} \sum_{i=1}^{K} \hat{f}(\mathcal{S}_i) + \alpha \sum_{i=1}^{K} \sum_{j=1}^{K} \text{sim}(\mathcal{S}_i, \mathcal{S}_j), \tag{3.12}$$

where $\{\mathcal{S}_i\}_{i=1}^{M}$ are the M qualified strategies filtered by the simulator score. α denotes a trade-off coefficient and sim is a pairwise similarity function between strategies. Solving Eq. 3.12 helps deliver a set of candidates that trade off between low predicted runtime (exploitation) and low similarity (exploration). The above problem is a

typical problem of submodular minimization, and we resort to the greedy algorithm for an approximate solution [109].

Since we do not have a continuous representation of strategies, we design two approaches to estimate the similarities between strategies. The first approach estimates the similarity of the two strategies $\mathcal{S}_1, \mathcal{S}_2$ by comparing each sub strategy $s_{i1} \in \mathcal{S}_1$ and $s_{i2} \in \mathcal{S}_1$ corresponding to the variable v_i, counting how many choices of each synchronization-affecting factor are the same, and use the final count as a measure of their similarity (higher is more similar). The second approach takes the cosine similarity between the hidden outputs of the simulator, whose inputs are $\mathcal{S}_1$ and $\mathcal{S}_2$, as a proxy of the strategy similarity. Empirically, we find the two approaches perform similarly, but the first similarity function does not depend on the simulator thus can be used when no trained simulator is available (e.g. for the baseline `AutoSync(-s)`).

3.4.3 Low-Cost Optimization using Transfer Learning

It is costly to perform end-to-end strategy optimization for a target domain $\mathcal{G}_t, \mathcal{D}_t$ on large clusters or large models from scratch. The cost is especially high if $\mathcal{G}_t$ will be only trained once (one-shot training). We consider transferring simulators trained on the data from a source domain $\mathcal{G}_s, \mathcal{D}_s$ for unseen $\mathcal{G}_t, \mathcal{D}_t$, which is advantageous if the source domain data already exists or any of the $\mathcal{G}_s$ and $\mathcal{D}_s$ are small so their trial run data is cheap to collect. The transferability is possible only because of our feature design – its predefined features are universal due to its domain-agnostic nature. The per-variable raw features are extracted from $\mathcal{G}$, $\mathcal{S}$, and $\mathcal{D}$, where $\mathcal{G}$ is the generic low-level dataflow graph representation and is capable of expressing all NN models, $\mathcal{S}$ is based on $\mathcal{D}$ and generated from a strategy space invariant to models as well. Hence, the feature components of different $(\mathcal{G}, \mathcal{D}, \mathcal{S})_i$ are invariant. The only remaining

Table 3.1: Models we have experimented with. Their implementations are from the `tensorflow/models` repository. IC: image classification, MT: machine translation, LM: language modeling.

Model	Task	Training data	bs/gpu	#params
ResNet50 [112]	IC	ImageNet	32	23M
ResNet101 [112]	IC	ImageNet	32	45M
InceptionV3 [113]	IC	ImageNet	32	24M
VGG16 [89]	IC	ImageNet	32	138M
DenseNet121 [77]	IC	ImageNet	32	8M
Transformer [3]	MT	WMT'14 ende	5K	62M
NCF-dense [88]	CF	MovieLens-20Mx16x32	256	122M
NCF-sparse [88]	CF	MovieLens-20Mx16x32	256	122M
BERT-3L [5]	LM	Wiki & BookCorpus	32	11M
BERT-6L [5]	LM	Wiki & BookCorpus	32	36M
BERT-base [5]	LM	Wiki & BookCorpus	32	110M
BERT-large [5]	LM	Wiki & BookCorpus	8	340M

variability is the length of features, which results from the numbers of variables in different $\mathcal{G}$. However, these numbers are absorbed by models like GAT and RNN, which operate on inputs with variable lengths. §3.5.7 validates the effectiveness of the transferable feature representation.

3.5 Experiments and Results

We build strategies using TensorFlow 2.0 and use it for distributed execution [110]. We conducted experiments on 10 models from different DL families, and 14 cluster configurations based on our in-house cluster and AWS. We provide additional details about the models and cluster specifications below and in Table 3.1 and Table 3.2 [3].

[3] In this chapter, we do not consider synchronization-affecting factors that would alter the algorithm or result as in the original single-node code, such as lossy compression [40], staleness [111], **etc.**

3.5.1 Models

Table 3.1 list the details of the 11 models we have experimented with, including 5 CNN models for image classification (IC), a transformer-based model for neural machine translation (MT), two versions of the neural collaborative filtering model for recommendation systems, and four BERT variants for language modeling. Their single-node implementations are from the official open source repository at `https://github.com/tensorflow/models`. Their training data, and number of parameters are listed in Table 3.1. We use default training settings suggested by MLPerf [114]. The per-GPU batch size is indicated in Table 3.1 as well.

It is worth noting that, for the neural collaborative filtering (NCF) model, we follow MLPerf [114] and use an enlarged version (x16x32) of the model and training set – which expands the original MovieLens-20M dataset [115] with 16x more users and 32x more movies. Details can be found at `https://github.com/mlperf/training/tree/master/recommendation/pytorch#getting-the-expanded-dataset`. To test the system's capability, we also experiment with two versions of NCF: NCF-dense, which uses dense tensors to represent the gradients of embedding variables in the model, and NCF-sparse, which uses sparse tensors (*IndexSlices* in TensorFlow) to represent the gradients instead.

3.5.2 Clusters

We focus on GPU clusters since it is the main setup for distributed DL training. During our research, we have tested on 14 different cluster configurations as listed in Table 3.2: *Cluster A* includes 11 nodes, each equipped with a GeForce TITAN X GPU, an Intel 16-core CPU and 64GB RAM, interconnected via a 40-Gigabit Ethernet switch; *Cluster B*, based on AWS, consists of up to 8 node, each node is one

of the g3.4xlarge (1x Tesla M60 GPU and 10GbE Ethernet), g3.16xlarge (4x M60, 25GbE), g4dn.2xlarge (1x NVIDIA T4 GPU and 25GbE Ethernet), g4dn.12xlarge (4x T4, 50GbE) instance types. Details about the configurations of AWS instances can be found at `https://aws.amazon.com/ec2/instance-types/g4/` and `https://aws.amazon.com/ec2/instance-types/g3/`. Due to AWS constraints, they all have 10GbE single-flow bandwidth. On top of these two clusters, we list all the resource specifications we have experimented with, including cluster setup, number of GPUs on each cluster node, and bandwidth information, in Table 3.2.

3.5.3 Baselines

As external baselines, we provide two robust hand-optimized synchronization systems. (1) `Horovod` [29]: This algorithm employs AllReduce (AllGather) to synchronize dense (sparse) gradients of all model variables and BO to autotune collective operation fusion. (2) `PS`: A manually optimized PS that incorporates several optimizations [116, 30, 31, 27], such as maintaining load balance across servers by partitioning and placement [116], autotuning credit size [27], and reducing network transfer via hierarchical reduction, *etc.* We employ them as collective- and PS-based baselines, respectively. It should be noted that both techniques employ the same TF and NCCL version for distributed execution and communication with AutoSync, preventing system-induced variations. We gather around 10,000 data points containing $(\mathcal{G}, \mathcal{D}, \mathcal{S})$ input tuples and associated ground truth runtimes on 11 models and 14 clusters.

3.5.4 Public Datasets

We have collected a dataset containing nearly 10k data points of $\{(\mathcal{G}, \mathcal{D}, \mathcal{S}), f\}$, where $\mathcal{G}$ is one of the DL models in Table 3.1, $\mathcal{D}$ is one of the cluster setups from Table 3.2,

Table 3.2: Cluster specifications we have experimented with, listed with their reference name ($\mathcal{D}$), setup information (Setup), GPU distribution and bandwidth specifications (Bandwidth spec).

$\mathcal{D}$	**Setup**	**GPU distribution**	**Bandwidth spec**
A1	16x Cluster A nodes	[1] x 16	40GbE
A2	11x Cluster A nodes	[1] x 11	40GbE
B1	2x g3.16	[4] x 2	25 GbE
B2	3x g3.16	[4] x 3	25 GbE
B3	4x g3.16	[4] x 4	25 GbE
B4	1x g4dn.12	[4] x 1	50 GbE
B5	2x g4dn.12	[4] x 2	50 GbE
B6	3x g4dn.12	[4] x 3	50 GbE
B7	4x g4dn.12	[4] x 4	50 GbE
B8	8x g4dn.12	[4] x 8	50 GbE
B9	1x g3.4, 1x g3.16	[1, 4]	10/25 GbE
B10	1x g3.16, 1x g4dn.12	[4] x 2	25/50 GbE
B11	2x g3.16, 2x g4dn.12	[4] x 4	25/50 GbE
B12	1x g4dn.2, 1x g4dn.12	[1, 4]	25/50 GbE

$\mathcal{S}$ is randomly sampled strategy from the proposed strategy space, and f is the groundtruth runtime collected via real distributed execution.

The dataset contains strategies sampled for all 11 models, ranging from fixed-formed strategies such as those used in specialized systems, and randomly explored strategies by AutoSync during the strategy auto-optimization. The data are organized into multiple folders where each folder corresponds to a domain of $(\mathcal{G}, \mathcal{D})$.

For quick access, we have provided scripts that read $\mathcal{G}$ as dataflow graphs in standard TensorFlow 2.0 format, and read the strategies and runtime into json formats. Instructions on how to access the dataset are provided at `https://github.com/petuum/autodist`.

Table 3.3: Comparisons of different model instantiations of the simulator.

Setting	Linear	RNN	GAT
NCF-dense, A	0.771	**0.894**	0.810
NCF-dense, B	0.826	**0.913**	0.830
VGG-16, A	**0.868**	0.796	0.753
VGG-16, B	0.833	**0.839**	0.692
BERT-base, A	0.758	0.746	**0.775**
BERT-base, B	0.807	**0.867**	0.760
BERT-large, A	0.780	**0.847**	0.771
BERT-large, B	**0.796**	0.755	0.784

3.5.5 Simulator Training Dataset and Hyperparameters

For the end-to-end results, we train simulators using runtime data collected on-the-fly during trials. For the ablation studies (Table.1-3), we split pre-collected data at auto-optimization of a specific domain (e.g., (NCF-dense, A)) into train/val/test at 70%/15%/15%, respectively, and report the ranking accuracy on the test split (averaged over 3 runs). Training RNN simulators in all settings use Adam with 1e-3 lr, decayed by 0.3 at the 80th/160th epoch, for 200 epochs. We clipped the gradient norm to 0.25.

3.5.6 End-to-End Results and Ablation Studies

Comparing Model Instantiations

To compare the linear model, RNN and GAT, we create datasets from trial data gathered in six different settings and train them as simulators, respectively. Table 3.3 shows their ranking accuracy on held-out test sets (the standard deviation of the results is within $\pm 2\%$ over three runs). RNN outperforms the linear model in most cases by using raw information. GAT, although also modeling the graph structure of $\mathcal{G}$, does not show significant advantages. We hypothesize that GAT may require more

Table 3.4: The per-iteration time statistics of 200 strategies found by RS and GA on (VGG16, A).

Stats	`RS`	`GA`
Mean (s)	1.07	0.67
Std. (s)	0.63	0.03
Min. (s)	**0.60**	0.64
Max. (s)	2.34	0.79

data for training that are currently out of reach due to our budgeted search. As a result, RNN is used by default in the rest of the chapter.

Search Algorithm Comparisons

On optimizing strategies for (VGG-16, A) with a budget of 200 trials, we compare two search algorithms: random search (RS) and GA. Table 3.4 displays the statistics for the per-iteration time of the 200 strategies discovered. GA, by definition, has a higher average quality and lower variane than RS, but it has a tendency to overfit with the simulator's (inaccurate) forecast, resulting in worse quality on the best discovered strategy. We, therefore, prefer RS over GA as the default algorithm for AutoSync strategy optimization.

Auto-Optimization Results

We employ AutoSync to optimize the strategies for NCF-dense (122M), VGG16 (138M), and BERT-large (340M). They span three different NN families but are all seen as "difficult-to-parallelize" workloads due to their ¿100M parameters. We contrast two AutoSync variants with their respective external baselines. (1) `AutoSync(-s)`, in which the simulator is disabled while searching, investigates 30K strategies at random and picks 200 candidates with the least amount of similarity(§3.4.2). (2) `AutoSync`: The full AutoSync with a budget of 200 for real evaluation on clusters.

To obtain the runtime f, we run 10 warm-up iterations, followed by 40 iterations of training, the runtime of which is averaged as the ground truth. Figure 3.2 contrasts the best-discovered strategy in 200 trials using two variants with the two manually optimized baselines. In four out of six settings and without a simulator, `AutoSync(-s)` discovers strategies up to 1.4x faster than the best one in `PS` and `Horovod`. `AutoSync` uses the simulator to find strategies 1.2x–1.6x quicker than baselines. To understand the speedup, BERT-large requires 2M steps [117] of training to achieve the observed accuracy with batch size 128 (batch size 8 on 16 GPUs). On cluster B, a 1.2x speedup decreases the training time by seven days and saves approximately $2200 AWS credits per training job. Furthermore, when applied to unseen data, a model must, but a trained simulator may be utilized repeated across jobs. When comparing `AutoSync` to `AutoSync(-s)`, the simulator, in addition to providing higher quality, drives the search to solutions faster, *e.g.*, on (BERT-large, A), `AutoSync` locates strategies 1.25x faster than `PS` in about 50 trials and 1.2x using only 30 trials on cluster B. Similar patterns have been observed in other settings.

Search Space Evaluation

We define the metric hit rate for each auto-optimization run as percentage of strategies examined up to trial k that are above hand-optimized baselines for each auto-optimization run. We compute the hit rates individually for each run and show the average percentage at k of three runs in the right axis of Figure 3.2. Except for (NCF-dense, B), `AutoSync(-s)` provides positive hit rates, and the rate for (VGG16, B) (42.0%) and (BERT-large, B) (28.5%) is acceptable. When used with a simulator, `AutoSync` typically achieves superior strategies, particularly on more complex models such as VGG16 and BERT-large ($> 70\%$). This evaluation demonstrates that factoring the strategy in relation to each variable and co-optimizing many factors generate

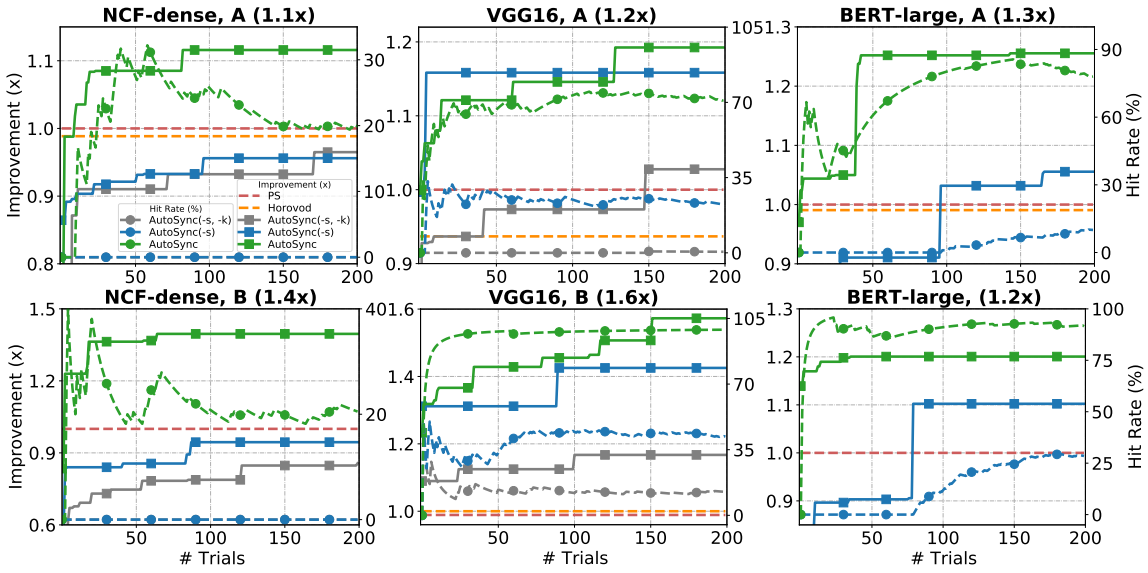


Figure 3.2: Comparing `AutoSync`, `AutoSync(-s)`, `AutoSync(-s,-k)` on (left axis) the improvement (higher is better) of the best found strategy over baseline, (right axis) the percentage of strategies better than baseline (higher is better), w.r.t. the number of trials conducted in 200 trials. The baseline (1x) is the better one of `PS` and `Horovod`). A curve is skipped from the plot if it is too below the baseline. The average over three runs is reported, the std of hit rates are in $\pm 3\% - 7\%$ and that of improvements over baseline at trial 200 are in $\pm .04 - .1$.

a promising space with a comprehensive set of strategies that outperforms manually optimized ones.

Knowledge Constraints

Figure 3.2 shows `AutoSync(-s,-k)`, a variation in which the knowledge constraints in §3.4.2 are removed from `AutoSync(-s)`. `AutoSync(-s,-k)` typically visits strategies that are below hand-optimized baselines, especially on complex models with a wider search space. For instance, on BERT-large, all strategies by `AutoSync(-s,-k)` are far below baselines; hence, they are skipped in the plots. It is thus crucial to incorporate system design principles as knowledge in order to make the search manageable.

Table 3.5: Studies on feature importance (pairwise ranking accuracy is reported). The std is within ±2% (3 runs).

Setting	Full	Raw only	Predefined only
NCF-dense, A	**0.894**	**0.894**	0.883
NCF-dense, B	**0.913**	0.907	0.873
VGG16, A	**0.796**	0.785	0.734
VGG16, B	**0.839**	0.837	0.816
BERT-large, A	0.847	0.848	**0.850**
BERT-large, B	**0.755**	0.746	0.735

Feature Importance

Table 3.5 shows the individual effects of $\boldsymbol{z}^{raw}$ and $\boldsymbol{z}^{pre}$ ablation. More specifically, we train RNN simulators with: (1) only predefined features $\boldsymbol{z}^{pre}$, (2) only raw features $\boldsymbol{z}^{raw}$, and (3) the full features, under 6 $(\mathcal{G}, \mathcal{D})$ settings. Then, we compare their ranking accuracy on test sets. The use of the full features yields the highest accuracy, illustrating that the predefined and raw features may include complementing information. Predefined features, on the other hand, can be useful in the early stages of a search since it is possible to directly rank strategies by using them without any training.

3.5.7 Transferring Trained Simulators

Transferability Studies

In contrast to §3.5.6, we train RNN simulators using trial data from a source domain $\mathcal{G}_s, \mathcal{D}_s$ to rank strategies in unseen target domains $\mathcal{G}_t, \mathcal{D}_t$, without any additional training, and report the ranking accuracy in Table 3.6. In general, we noted several points. (1) Models with comparable architectures are more transferable. When $\mathcal{D}$ is constant, transformer-based models [3] transfer quite well with the lowest accuracy at 0.76. When $\mathcal{D}$ is changed, the transferability is slightly compromised owing to

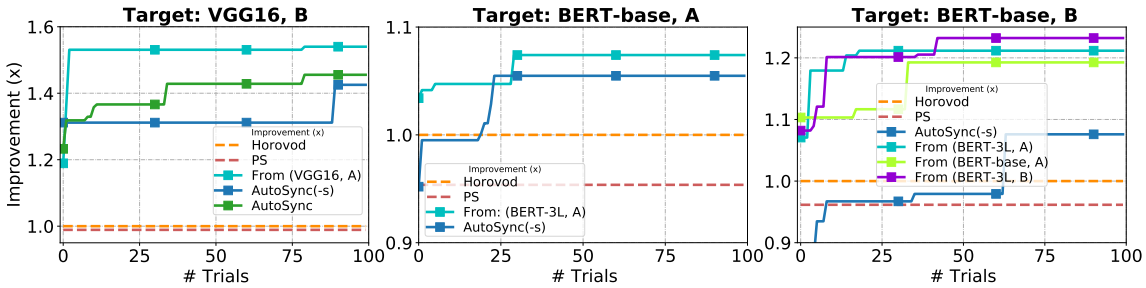


Figure 3.3: Transferring the trained simulators from different source domains to 3 target domains, compared to untransferred `AutoSync-` and `AutoSync` with a budget of 100 trials. The average of three runs is reported.

increasing domain distance. This difference suggests that we can optimize the strategy of a larger BERT model using a simulator trained for smaller BERT models. (2) Models with similar size distributions of variables show transferability. For example, despite having only 10 variables, the NCF-dense model can transfer to BERT-base with a decent accuracy of 0.7904; VGG-16, as a CNN, ranks strategies of BERT-base better (0.7298) than ResNet101 (0.56); and both VGG16 and BERT-base have uneven communication loads caused by several extremely large variables. (3) When $\mathcal{G}$ is fixed, transferability across $\mathcal{D}$ is seen. In practice, we may pre-train a simulator on low-cost in-house clusters (*e.g.*, cluster A), before deploying it for training jobs on more expensive and larger-scale clusters.

End-to-End Results on Transfer Learning

We are currently transferring trained simulators to help guide end-to-end optimizations. We purposefully choose models that are difficult to parallelize and clusters that are expensive: (BERT-base, A), (BERT-base, B), and (VGG16, B). We establish a smaller budget of 100 trials and do not use data from target domains for model selection or finetuning. The optimization progress is depicted in Figure 3.3. Surprisingly, a well-trained simulator from On (VGG16, A) can find identify 1.5x quicker in as few

Table 3.6: The target domain test accuracy under different transfer learning settings.

Source → Target	Accuracy
(BERT-3L, A) → (BERT-3L, B)	0.8672
(Transformer, A) → (BERT-base, A)	0.7674
(BERT-3L, A) → (BERT-base, A)	0.7591
(BERT-3L, B) → (BERT-base, B)	0.8100
(NCF-dense, A) → (BERT-base, B)	0.7904
(VGG16, A) → (BERT-base, B)	0.7298
(BERT-3L, A) → (BERT-base, B)	0.6992
(BERT-base, A) → (BERT-base, B)	0.6852
(VGG16, B) → (BERT-base, B)	0.6774
(Transformer, A) → (BERT-base, B)	0.6672
(BERT-3L, A) → (Transformer, A)	0.8866
(Transformer, B) → (Transformer, A)	0.8305
(Transformer, A) → (Transformer, B)	0.8171
(BERT-3L, A) → (Transformer, B)	0.808
(NCF-dense, A) → (NCF-dense, B)	0.7694
(VGG16, A) → (VGG16, B)	0.7966
(BERT-3L, A) → (VGG16, B)	0.5583
(ResNet50, A) → (ResNet101, A)	0.6057
(VGG16, A) → (ResNet101, A)	0.5600
(VGG16, A) → (ResNet50, A)	0.7156
(VGG16, A) → (DenseNet121, A)	0.7596
(ResNet101, A) → (ResNet101, B)	0.7187
(DenseNet121, A) → (ResNet50, B)	0.5266
(VGG16, A) → (InceptionV3, A)	0.7857

as five trials on (VGG16, B). On (BERT-base, B), we experiment with three different source domains and discover that the domain distance has an effect on the end-to-end results: source (BERT-3L, B) achieves higher quality and efficiency than source (BERT-base, A). We recommend a more in-depth examination of the transferability in future research. Overall, we are able to transfer simulators trained on "cheap" domains to uncover good strategies in "expensive" domains after only a few trials. Aside from reducing the number of trials, effectively transferring a simulator reduces the wall-clock time used in auto-optimization by bypassing simulator training, which is beneficial in cases where single-shot model training occurs often.

3.6 Conclusion

The proposed AutoSync facilitates a novel search space of synchronization-affecting factors and learns to optimize synchronization strategies at a variable-level in order to improve training performance. AutoSync can identify synchronization strategies up to 1.6x better than those manually optimized when guided by an ML-based simulator and proper knowledge, even with just 200 trial run training data for the simulator. A dataset including 10,000 strategy, model, cluster, and runtime tuples are made accessible with the proposed framework to aid future studies on automating the parallelization of ML programs.

Part II

Learning Universal Knowledge

Chapter 4

OPTIMUS

4.1 Introduction

The first part of the dissertation summarizes our efforts in developing effective and efficient Transformer-based architectures for learning expert knowledge, including the Graph Transformer that allows systematic injection of medical knowledge graphs containing disease and abnormality relationships into medical report generation, and the AutoSync framework that propitiously leverages a Transformer-based simulator guided by human-designed knowledge constraints for automatic and adaptable search of the synchronization strategies. In this part, we turn our attention from expert knowledge learning to its opposite: universal knowledge learning. Unlike expert knowledge, which is usually organized, structured, and summarized in each specific domain, universal knowledge is ubiquitous and does not follow any particular form, making it difficult to define, collect, and learn. Imagine a newborn baby who learns about the world by looking at tons of images, language, *etc.* The more she observes, the more she learns and connects with her existing cognition. For example, by looking at a few images of dogs without anyone telling her any definition of dogs, she can immediately tell whether a new image contains dogs or not. This mechanism is later formalized as SSL by the research community. In the second part of the dissertation, we investigate the ability of Transformer-based models combined with SSL learning methods to learn universal knowledge. Note that both the Transformer-based model architectures and the SSL approaches are key aspects of this part, as only a good

match between these two aspects can lead to good performance.

As one of the most typical SSL methods, pre-training language models (LMs) aim to learn universal language representations through web-scale unlabeled data and pretext tasks. Recently, various influential pre-trained models [5, 118, 6, 7, 119, 120, 8, 11, 32, 33] have been shown to be effective for a variety of downstream NLP tasks. In general, there are two categories of pre-trained LMs: autoencoding and autoregressive. Autoencoding models focus on the encoding process. Within this category, BERT [5] proposes masked language modeling (MLM) and next-sentence prediction (NSP) objectives. RoBERTa [9] examines the impact of many key hyperparameters and training data sizes adopted in BERT and subsequently improves its robustness. Albert [10] is a lightweight BERT that applies factorized embedding parameterization and cross-layer parameter sharing techniques to reduce memory consumption and speed up the training of BERT. In the autoregressive category, GPT [6] first proposes using a generative objective to pre-train the Transformer. The subsequent GPT-2 [7] and GPT-3 [33] increase the model and data size, obtaining improved performance when transferring to various downstream NLP tasks. XLNet [11] proposes learning bidirectional contexts by maximizing the expected likelihood over all permutations of the factorization order. Several model variants have been proposed to combine language understanding and generation tasks in a unified framework, including UniLM [34], BART [13], and T5 [35]. Even though these models substantially advanced the state-of-the-at across a wide range of NLP tasks, they fail to explicitly model the structures in a latent space, thereby making it difficult to control natural language generation/representation from an abstract level.

VAEs [121, 122] appear to be the most suitable solution for integrating language understanding and generation tasks in a unified framework, as they connect an encoder and a decoder via a latent space. With effective training of the latent space, vectors

in the space can represent the global features of any sentence. Therefore, by slightly manipulating the vectors, we can orderly control the language representation and generation [123, 124]. Moreover, due to the small size of the latent vectors compared with the raw input, smooth feature regularization and guided sentence generation can be easily performed. However, despite the appeal of the theory, existing VAEs are dominated by small networks, such as two-layer LSTMs [125], preventing the models from achieving optimal solutions and robustness.

To alleviate this problem, OPTIMUS [19]—a large-scale VAE implemented by a multilayer Transformer encoder and decoder—is proposed for controllable and scalable language representation learning. [1] OPTIMUS embraces sentence-level representation learning and generation and therefore learns a sentence-level latent space. Moreover, by integrating both an encoder and a decoder, OPTIMUS not only inherits the advantages of both BERT and GPT models by being capable of performing both language understanding and generation tasks, but also improves the previous models by learning a structured and ordered latent semantic space to enhance both universal representation learning and controllable generation. Extensive experiments are conducted to examine the effectiveness of the proposed OPTIMUS, including language modeling, dialog response generation, text style transfer, and low-resource language understanding. It is observed that OPTIMUS achieves lower perplexity than GPT-2 on standard benchmarks, produces strong performance on guided text generation, and improves BERT on feature-based language understanding tasks.

[1] The author of the dissertation is the third author of the paper [19]. Her major contribution to this work is transferring the pre-trained OPTIMUS to the downstream label-conditional text generation task.

4.2 Related Work

4.2.1 Language Models (LMs)

Skip-thought models [126] utilize continuous text from books to train an encoder-decoder architecture using a reconstruction loss. Standard sequence autoencoders [127] adopt two approaches to enhance sequence learning: a conventional language model that predicts the next word given a sequence and a sequence autoencoder that reconstructs input sequences by projecting input into a latent representation space. Later works adopted the Transformer model for language modeling, such as BERT [128], which reconstructs missing words given contexts by employing a Transformer encoder, and GPTs [6, 7, 33] that predicts the next words using a Transformer decoder. Inspired by the success of BERT and GPTs, there has been a surge in related research, such as RoBERTa [9], Albert [10], XLNet [11], UniLM [34], BART [13], and T5 [35]. Despite state-of-the-art performance in various downstream NLP tasks, these models mostly lack explicit and structured modeling for the latent semantic space, making controllable generation, such as label-conditional text generation [4] and style transfer [129], difficult.

4.2.2 Linguistic VAEs

VAEs possess many technical merits for learning generic language representation, such as increased generation diversity and a learned structured latent semantic space that allows effective manipulation and controllable generation. However, to achieve practical performance improvement, the KL vanishing problem in VAE training must be be properly addressed. Various strategies and methods have been proposed to alleviate this problem, including levering different KL annealing or thresholding

strategies [123, 130, 131, 132], decoder architectures [133, 134], auxiliary loss [135], semi-amortized inference [136], aggressive encoder training schedule [137], and flexible posterior [138]. However, while these methods are applicable to small networks, such as 2-layer LSTMs, they fail to offer a scalable solution for large-scale VAEs.

4.3 Preliminaries

We first introduce OPTIMUS [19], which exploits a Transformer-based VAE model for controllable language representation learning. Then, we present transfer learning for label-conditional text generation, employing a Transformer-based model architecture, and OPTIMUS's pre-trained weights.

OPTIMUS embraces a Transformer encoder-decoder architecture optimized by a reconstruction loss and a latent space optimized by a KL divergence [19]. OPTIMUS encodes a given input sentence into a vector in the latent space, regularizes the vector by approaching a prior distribution, and triggers the decoder to reconstruct the input sentence based on the latent vector. Specifically, the marginal log likelihood is written as follows:

$$\log p_\theta(\boldsymbol{x}) = \log \int p(\boldsymbol{z}) p_\theta(\boldsymbol{x}|\boldsymbol{z}) dz, \tag{4.1}$$

where θ represents the model parameters, $\boldsymbol{x}$ denotes the input sentence, and $\boldsymbol{z}$ is the latent vector [19]. Due to the numerous possible decoder outputs, solving this equation is intractable, and variational inference is required by approximating the true posterior $p_\theta(p(\boldsymbol{z})|\boldsymbol{x})$ using the variational distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ [19]. The mathematical

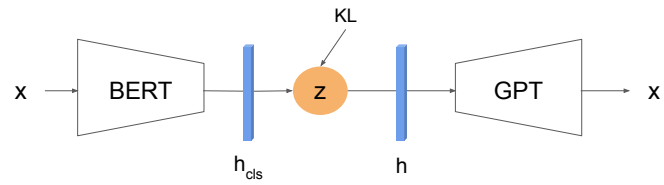


Figure 4.1: Illustration of OPTIMUS architecture.

equation is written as follows:

$$\log p(\boldsymbol{x}) \geq \mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p(\boldsymbol{x}|\boldsymbol{z})\big] - \text{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})), \tag{4.2}$$

where ELBO is the evidence lower bound objective. Usually, $p(\boldsymbol{z})$ is a multivariate Gaussian distribution [19].

When generating a novel sentence, OPTIMUS first randomly samples a vector z from the latent space with prior distribution $p(\boldsymbol{z})$, and then projects the latent vector to a sentence y in the output space using the decoder module [19]. The generation process can be written as follows:

$$p(\boldsymbol{y}|\boldsymbol{z}) = \prod_{t=1}^{T} p(x_t|x_{<t}, \boldsymbol{z}). \tag{4.3}$$

The encoder and decoder are implemented using a multilayer Transformer encoder and decoder, respectively. The detailed model architecture is illustrated in Figure 4.1. In addition to the novel Transformer-based VAE architecture, several significant design choices are adopted to simplify the model's complexity and improve performance, including:

- initializing OPTIMUS with BERT and GPT's pre-trained weights,
- implementing the encoder as a $\text{BERT}_{\text{base}}$ and implementing the decoder as a GPT-2 containing 12 layers,

- employing different tokenization schemes for the encoder and decoder (*e.g.*, WordPiece embedding (WPE) for the encoder and Byte Pair Encoding (BPE) [7] for the decoder),
- and annealing the weight of the KL divergence.

4.4 Label-Conditional Text Generation

The goal of this task is to generate sentences conditioned on a given label. We consider a two-stage algorithm to adapt OPTIMUS for this task. First, we fine-tune a VAE language model on the downstream dataset, and freeze the model parameters. In other words, the latent space is fixed. Second, we build a conditional GAN for the latent space. Let's denote the latent vectors for ground-truth sentences as $\boldsymbol{z}_{\text{true}}$. We build a generator G to produce $\boldsymbol{z}_{\text{fake}} = G(\epsilon, y)$, where ϵ is the random noise, and y is the label. A discriminator D is simultaneously trained to distinguish $\boldsymbol{z}_{\text{true}}$ and $\boldsymbol{z}_{\text{fake}}$ sentences. The learning objectives for conditional GAN are as follows:

$$\begin{aligned}
&\min_{G} \max_{D} \mathcal{L}_{\text{cGAN}} \\
&= \mathbb{E}_{\boldsymbol{x}, y \sim q(\boldsymbol{x}, y)} \big[\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x})} [\log p_D(d = 1 | E(\boldsymbol{x}))] \\
&+ \mathbb{E}_{\epsilon \sim p_0(\epsilon)} [\log p_D(d = 0 | G(\epsilon, y))] \big]
\end{aligned} \tag{4.4}$$

To make the model work effectively, learning a smooth and meaningful latent space for target sentences is key. The text generation procedure conditioned on label y is as follows:

$$\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{z}), \text{ with } \boldsymbol{z} = G(\epsilon, y) \tag{4.5}$$

Table 4.1: Label-conditional text generation on Yelp dataset.

Metrics	Ctrl-Gen	ARAE	NN-Outlines	OPTIMUS OPTIMUS
Accuracy↑	0.878	0.967	0.553	**0.998**
Bleu ↑	0.389	0.201	0.198	**0.398**
G-score↑	0.584	0.442	0.331	**0.630**
Self-Bleu↓	0.412	0.258	0.347	**0.243**

This mimics the process of producing sentence outlines using conditional GAN and filling in details using the decoder.

4.5 Experiments and Results

4.5.1 Baselines

We compare with three baselines: (1) *Ctrl-Gen* [124]; We use their released code to reproduce the results. (2) *ARAE* [139] proposes to learn an auto-encoder first and then train a GAN to produce the latent vectors. (3) *NN-Outlines* [140] proposes the use of a general-purpose encoder for text generation, and we implement it using BERT. Note that our two-stage, fine-tuning scheme borrows ideas from ARAE and NN-Outlines. The key difference is that we employ our pre-trained OPTIMUS model and work on a better latent space.

4.5.2 Evaluation

We consider three metrics: (1) the *Bleu* for sentence quality, (2) the *Accuracy* for conditional generation capability to correctly predict the attributes that generated sentences are conditioned on, assessed by an oracle classifier, and (3) the *G-score* is reported as the geometric mean of Accuracy and Bleu. This is the most important metric because it evaluates overall performance. For label-conditional text generation,

Positive
our favorite place to get great coffee and taterts.
the best brunch you will find in vegas.
the best breakfast with meats is awesome!
great samosas and serve as a regular!
great place to meet up with a custom bean & wine.
a great selection of chinese food and always happy.
the free wi-fi is amazing as well!
great staff and freshly made latte is a must.
love the fresh staff as well!
highly recommend the place and sunbeams!
the staff is always great with homemade paesadillas.

Negative
not only did you get a headache upstairs, they were disgusting.
once i realized the pizza wasn't decent, i cancelled.
instead of going to the bathroom you couldn't find anything.
tonight i was unable to give the pizza any less.
i didn't even bother to find a $ [num] frozen pizza.
no wonder i was dropped off at laundry.
not only was this place freezing, but the salad sucked.
then [num] bucks was ruined in my mouth.
love the fresh staff as well!
once you asked for chipotle its out of control.
another thing i refused to eat.

Figure 4.2: Label-conditional text generation on Yelp dataset. The top block shows the positive reviews, and bottom block shows the negative reviews.

the Bleu of each generated sentence is computed by comparing it with all the sentences in the test set, as there are no source sentences. We further report Self-Bleu [141] to evaluate the diversity of the generated sentences.

4.5.3 Dataset

We consider the short Yelp dataset collected in [142]. It contains 444K training sentences, and we use separated datasets of 10K sentences for validation/testing, respectively. The goal is to generate text reviews given the positive/negative sentiment.

4.5.4 Results and Analysis

The results are shown in Table 4.1, OPTIMUS achieves the best performance on all metrics. The conditional-generated sentences are shown in Figure 4.2.

4.6 Conclusion

We leveraged OPTIMUS—a large-scale pre-training VAE—for downstream label-conditional text generation task. Because OPTIMUS learned a smooth and structured latent space, we observed significant performance improvement over competitive baselines. The visualization also demonstrates the effectiveness of VAE for controllable text generation.

Chapter 5

Complementary Auxiliary Classifiers

5.1 Introduction

In Chapter 4, we study pre-training a large-scale VAE model for language modeling and transferring it to the downstream label-conditional text generation task. The amalgamation of the superior VAE architecture employing multilayer Transformer encoder-decoder and the pre-trained weights of OPTIMUS obtained from pre-training on massive amount of text corpus have proved to be effective for efficiently acquiring knowledge contained in the downstream task and leading to significant performance boosts. In this chapter, we continue exploring expressiveness of the Transformers for label-conditional text generation task by establishing a generative adversarial network (GAN) that enhances the non-identifiability issue of the previous methods.

Text generation is an important challenge in natural language processing (NLP). Most previous research in this area has focused on unsupervised text generation [143, 144], and success has recently been achieved with pre-trained generative training [7, 11]. However, while these setups measure the ability of models to generate the coherent content of a sentence, they do not address more natural human communication in a given context. For example, generating more engaging conversions requires conditioning on personality in image captioning and dialogue systems. The synthesis of coherent sentences requires conditioning on a given topic/sentiment. These setups can be formulated as label-conditional text generation.

Several attempts have been made to solve this problem, and most are based on

deep latent variable models (LVMs), such as variational autoencoders (VAE) [121] and their conditional variants [124, 133]. One attractive property of these models is that they map sentences to global language features in a latent space and allow the manipulation of generated sentences with a specific tense, sentiment, or topic.

One prominent challenge for LVMs is to learn smooth and disentangled latent representations such that generation from this space results in realistic sentences that can be effectively classified during conditional decoding. Hu *et al.* [124] proposed Ctrl-Gen, which uses a VAE to represent sentences as smooth Gaussian distributions regularized toward a standard normal prior distribution in the latent space. However, VAE-based methods [121] are difficult to train due to their notorious posterior collapse and KL vanishing [123]. To solve this problem, ARAE [145] uses adversarial learning to construct a flexible prior distribution and further proposes an auxiliary classifier in the latent space to disentangle the learned latent feature from the conditional labels.

In this chapter, we reexamine the relationship between disentangled feature learning and label-conditional generation and show that the former does not necessarily lead to the latter. We term this issue a non-identifiability issue. This is manifested when the generator is able to resemble training samples perfectly but degenerates to ignoring the conditional label and solely relying on the latent code, even though the encoder learns label-agnostic representations.

To solve this issue, we propose the Complementary Auxiliary Classifier Regularized Auto-encoder (CARA), which has are two classifiers to predict labels: a classifier in the latent space is used for disentangling feature learning, as in ARAE, and a complementary classifier in the observation space is proposed to encourage the generator to contain the conditional label. We prove that the proposed complementary auxiliary classifiers introduce a cycle-consistency loss for conditional labels, leading to maximizing the mutual information between the generated sentences and their

corresponding labels. Moreover, we explore the trade-off between generation quality and label-conditional accuracy and propose two practical techniques for improving overall performance: (*i*) an annealing training schedule for the proposed complementary auxiliary classifier to gradually incorporate label-conditional supervision signals, and (*ii*) BERT [5] is used as an encoder to provide more universal and generalizable latent representations. We show that this approach provides empirically improved LVM-based conditional text generation.

We built a suite of four datasets for the comprehensive study of label-conditional text generation. The quantitative and qualitative experimental results demonstrate that the proposed techniques consistently show improved performance. We further apply CARA to the style transfer task, in which CARA achieves state-of-the-art performance.

5.2 Related Work

5.2.1 Difference from Attribute Transfer

This chapter focuses on label-conditional text generation, where $\boldsymbol{z}$ is drawn from a latent distribution, allowing generation of diverse sentences. This task is different from attribute transfer, where $\boldsymbol{z}$ is extracted from a given sentence and fixed during sentence generation. Therefore, label-conditional text generation requires learning a smooth latent space, such that sampling from this space leads to faithful linguistic sentences. Many methods of attribute transfer treat the problem as sequence-to-sequence translation, and source sentences must be provided in order to transfer labels in target sentences [146, 147, 148]. Such methods are not able to conduct label-conditional generation due to the lack of a smooth space from which to draw

samples. In contrast, the proposed CARA can be used for label-conditional generation and style transfer tasks.

5.2.2 Differences from Prior Work

The work most related to ours is Ctrl-Gen [124], ARAE [145] and NN-Outlines [140]. In Ctrl-Gen, a label classifier is adopted into the observation space. However, our model learns a smoother latent space via adversarial learning, and a more effective disentanglement constraint is enforced by the auxiliary classifier in the latent space. ARAE leverages separate generators for each conditional label and trains them only with samples within the corresponding category. This aims to implicitly alleviate the non-identifiability issue but sacrifices performance, as each generator is trained with fewer samples. It is also inherently impractical when the number of labels increases. On the contrary, CARA does not suffer from either performance degradation due to fewer samples or excessive parameters. NN-Outlines uses a pre-trained general purpose sentence encoder to provide black-box high-level "outlines", and a generative adversarial network in the latent space to match the distribution of the latent representations induced by the encoder. Compared to CARA, NN-Outlines does not have any disentanglement constraints, or additional label control supervision signals.

5.3 Preliminaries

Consider a training set $\mathcal{S} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ of pairwise data, where $\boldsymbol{x}_n = [x_{n,1}, x_{n,2}, ..., x_{n,T_n}]$ is a text sequence of length T_n, and $y_n \in \mathcal{Y}$ is its corresponding label/attribute/class. The goal of conditional text generation is to generate a new $\boldsymbol{x}$ for a given y. Due to the diversity of natural language, a latent code $\boldsymbol{z}$ is introduced to characterize diverse information associated with y, which is required to generate faithful and variable $\boldsymbol{x}$.

Typically, $\boldsymbol{z}$ is drawn from an easily sampled prior distribution $p(\boldsymbol{z})$, such as Gaussian $\mathcal{N}(0, \mathbf{I})$.

Given $\boldsymbol{z}$ and y, the sampling procedure for the conditional $p_G(\boldsymbol{x}|\boldsymbol{z}, y) = \prod_{t=1}^{T} p_G(x_t|\boldsymbol{x}_{<t}, \boldsymbol{z}, y)$ is performed in a sequential manner, and an auto-regressive generator model G is used to generate x_t at the every time step:

$$x_t = G(\boldsymbol{x}_{<t}, \boldsymbol{z}, y), \tag{5.1}$$

where $\boldsymbol{x}_{<t}$ indicates all tokens before t. The synthesis of a given sentence continues until the end-of-sentence symbol is manifested.

5.3.1 Autoencoder for Conditional Text Generation

The generator G is typically learned by maximizing the marginal log likelihood $\log p_G(\boldsymbol{x}) = \log \int p_G(\boldsymbol{x}|\boldsymbol{z}, y) p(\boldsymbol{z}) p(y) \mathrm{d}\boldsymbol{z}\mathrm{d}y$, where $p(y)$ is the label distribution. However, the marginalizing integral wrt $\boldsymbol{z}$ is intractable to compute for many generator choices. Thus, variational inference is considered, and the true posterior $p_G(\boldsymbol{z}|\boldsymbol{x}) \propto p_G(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z})$ is approximated via the variational distribution $q_E(\boldsymbol{z}|\boldsymbol{x})$, implemented via an encoder:

$$\boldsymbol{z} = E(\boldsymbol{x}, \varsigma), \varsigma \sim q_0(\varsigma), \tag{5.2}$$

where $q_0(\varsigma)$ is the noise distribution. The observed sentence $\boldsymbol{x}$ can be represented using $\tilde{\boldsymbol{z}} \sim q_E(\boldsymbol{z}|\boldsymbol{x})$, and reconstructed as $\hat{\boldsymbol{x}} \sim p_G(\boldsymbol{x}|\tilde{\boldsymbol{z}}, y)$, where y is the corresponding label of $\boldsymbol{x}$. A reconstruction loss is applied to the generated sentence and observation:

$$\min_{E,G} \mathcal{L}_{\text{rec}} = -\mathbb{E}_{\tilde{\boldsymbol{z}} \sim q_E(\boldsymbol{z}|\boldsymbol{x}), y \sim p(y)} [\log p_G(\boldsymbol{x}|\tilde{\boldsymbol{z}}, y)]. \tag{5.3}$$

There are challenges in optimizing the auto-encoder objective in (5.3) for conditional text generation.

- **Smoothness**: The learned latent code $\boldsymbol{z}$ should be a smooth distribution such that samples or interposition of samples from it can lead to plausible sentence generation. This problem is inherited from using an auto-encoder for generic sentence generation.

- **Disentanglement**: For better control of attributes, $\boldsymbol{z}$ and y are desired to be disentangled, where the features encoded in each variable are exclusive of each other. Independence constraints have been applied to these two variables in prior work, such as reconstruction of $\boldsymbol{z}$ through an encoder [124] and using a latent space discriminator to enforce $\boldsymbol{z}$ to be independent from y [145].

5.3.2 Adversarially Regularized Autoencoders

Various techniques have been proposed to learn smooth and disentangled latent representations [131, 149, 150]. In the context of conditional text generation, adversarially regularized autoencoders (ARAEs) [135] have been introduced. Learning ARAEs can be viewed as proceeding in two alternating steps: *disentangled feature learning* and *adversarial feature generation*.

In the first step, in addition to the reconstruction objective in (5.3), the inferred $\boldsymbol{z}$ is trained to contain only information exclusive of $\boldsymbol{y}$. This disentanglement objective is achieved using an auxiliary classifier $C(\boldsymbol{z})$:

$$\min_{E} \max_{C} \mathcal{L}_{\text{disentangle}} = \mathbb{E}_{\boldsymbol{z} \sim q_E(\boldsymbol{z}|\boldsymbol{x})}[\log p_C(y|\boldsymbol{z})], \tag{5.4}$$

where the classifier C is learned to categorize $\boldsymbol{z}$ into its corresponding labels, while E

is forced to infer $\boldsymbol{z}$ to fake the classifier C. Ideally, at convergence, one cannot predict $\boldsymbol{y}$ using $\boldsymbol{z}$, and thus $\{\boldsymbol{y}, \boldsymbol{z}\}$ are disentangled.

In the second step, we learn a prior using a neural sampler $\boldsymbol{z} = S(\epsilon)$ to replace the fixed prior, where $\epsilon \sim p_0(\epsilon)$ is an auxiliary distribution from which one may easily draw samples. The neural sampler is trained to generate features to match the distribution $q(\boldsymbol{z})$ learned in the first step.

A discriminator $D(\boldsymbol{z})$ is introduced to distinguish the domain of $\boldsymbol{z}$, with $d = 1$ indicating $\boldsymbol{z}$ comes from $E(\boldsymbol{x})$ and $d = 0$ indicating $\boldsymbol{z}$ comes from $S(\epsilon)$. The min-max objective in the latent space can be written as follows:

$$\min_{E,S} \max_{D} \mathcal{L}_{\text{adversarial}} = \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}[\log p_D(d = 1|E(\boldsymbol{x}))] + \mathbb{E}_{\epsilon\sim p_0(\epsilon)}[\log p_D(d = 0|S(\epsilon))].$$

where $p(\boldsymbol{x})$ is the data distribution. When the optimum is achieved, the generated feature distribution $\pi(\boldsymbol{z})$ induced by $\boldsymbol{z} = S(\epsilon)$ can match the disentangled distribution $q_E(\boldsymbol{z}) = \int q_E(\boldsymbol{z}|\boldsymbol{x})q(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$.

The two steps are updated iteratively. Since (5.4) in the first step ensures the disentangled representation of $q(\boldsymbol{z})$ wrt to $p(y)$, and (**??**) in the second step ensures that the marginal distribution matches $\pi(\boldsymbol{z}) = q(\boldsymbol{z})$, the two steps together can guarantee that the generated features $\pi(\boldsymbol{z})$ from the neural sampler S can characterize the disentangled representation wrt to $p(y)$ at its optimum.

The full objective of ARAEs for the encoder E and generator G is:

$$\min_{E,G} \mathcal{L}_{\text{ARAE}} = \mathcal{L}_{\text{rec}} + \mathcal{L}_{\text{disentangle}} + \mathcal{L}_{\text{adversarial}}. \tag{5.5}$$

We see that ARAEs can be viewed as regularizing the basic auto-encoder objective in (5.3) with a disentanglement term and an adversarial term.

Advantages of ARAEs

ARAEs have the following characteristics: (*i*) they provide a smooth latent space for discrete sequences with a flexible learned prior. Compared to variational auto-encoder (VAE)-based methods [121], ARAEs do not suffer from KL vanishing or posterior collapse [123, 121, 151] caused by using KL-divergence with a fixed prior as regularization in the latent space; and (*ii*) the classifier C applied in the latent space explicitly constrains the encoder E to only record attribute-independent features, encouraging disentanglement of latent features and attributes during encoding, and thus improving control and manipulation of attributes during conditional generation performed by generator G.

5.4 Complementary Auxiliary Classifiers

5.4.1 Pitfalls of ARAEs

In ARAEs, the generator G learns to generate text $\boldsymbol{x}$ from a joint distribution constructed by the latent code $\boldsymbol{z}$ and the label condition y. During the disentangled feature learning stage of ARAE, the encoder E is trained to satisfy two objectives:

$$\min_{E} \mathcal{L}_{\text{rec}} + \mathcal{L}_{\text{disentangle}}. \tag{5.6}$$

This means that (*i*) the reconstruction $\mathcal{L}_{\text{rec}}$ in (5.3) encourages $\boldsymbol{z}$ to be unique in the latent space, so that the original $\boldsymbol{x}$ can be perfectly reconstructed, and (*ii*) the disentanglement $\mathcal{L}_{\text{disentangle}}$ in (5.4) encourages $\boldsymbol{z}$ to be disentangled from y, and thus y becomes the only source to control the label of the generated sentences. However, we argue that it is challenging for the encoder E to produce $\boldsymbol{z}$ that simultaneously

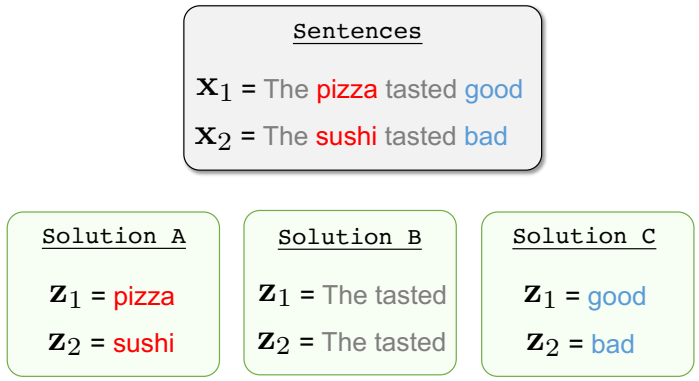


Figure 5.1: Illustration of possible solutions to the disentangled feature learning objective in (5.6). For the example sentences shown in the top row, three possible solutions are provided in the bottom row. Solutions A and B are valid to (5.6) and Solution C is false.

satisfies both objectives, and generator G to rely only on $\boldsymbol{z}$ (ignore the dependence of y) during generation.

5.4.2 Disentanglement versus Controllable Generation

The above implies that disentanglement does not necessarily lead to controllable generation. We call this phenomenon the non-identifiability issue in disentangled feature learning. This problem occurs only under the assumption that training samples are non-parallel, where there is no sentence pair that has the same content but the opposite labels. This is common in many cases, such as text style transfer and conditional generation with non-parallel data.

We constructed an example in Figure 5.1 to explain this further, where the dataset only contains two sentence samples: $\mathcal{S}_1 = \{(\boldsymbol{x}_1, y{=}1), (\boldsymbol{x}_2, y{=}0)\}$. Here, the conditioning information y is the sentiment, and the auxiliary classifier C learns to categorize "`good`" into $y{=}1$ and "`bad`" into $y{=}0$. We presented three potential candidate solutions. First, the disentanglement objective encourages the encoder to learn to summarize label-agnostic information, such as "`The pizza tasted`" and

"`The sushi tasted`" into $\boldsymbol{z}$, which is independent from the label-related information such as "`good`" and "`bad`". Therefore, Solutions A and B are valid, and Solution C is invalid with respect to the disentanglement objective. Furthermore, it can be shown that both Solutions A and B satisfy the reconstruction objective, as the joint $\{\boldsymbol{z}, y\}$ is unique enough to identify its corresponding $\boldsymbol{x}$. If Solution A is chosen, the model can perfectly generate $\mathcal{S}_1$ using $p_G(\boldsymbol{x}|\boldsymbol{z})$, rather than $p_G(\boldsymbol{x}|\boldsymbol{z}, y)$.

Ideally, the generator is expected to synthesize sentences successfully conditioned on any combination of label-agnostic and label-related information by training with the objectives in (5.5). However, as the dataset is non-parallel, the model only sees a limited combination of label-agnostic and label-related information. For example, in the illustrative dataset, the model is only able to see pairs of "`pizza`, `good`" and "`sushi`, `bad`" instead of a full set of pairs (*e.g.,* "`pizza`, `good`", "`pizza`, `bad`", "`sushi`, `good`" ,and "`sushi`, `bad`"). Thus, by only relying on one source such as "`pizza/sushi`" or "`good/bad`", the generator is able to perfectly reconstruct all training sentences. In this case, the generator can learn a degenerated distribution $p_G(\boldsymbol{x}|\boldsymbol{z})$ or $p_G(\boldsymbol{x}|y)$ instead of $p_G(\boldsymbol{x}|\boldsymbol{z}, y)$, depending on which source is dominant in the dataset.

5.4.3 Proposed Method: CARA

To solve this issue, we propose a **C***omplementary* **A***uxiliary classifier* **R***egularized* **A***uto-encoder* (CARA). It enhances the ARAE with an additional auxiliary classifier for the generated sentences. Specifically, during training, the generator synthesizes a sentence $\hat{\boldsymbol{x}}$ conditioned on a latent code $\boldsymbol{z}$ and a sampled conditional label y, which is then fed to a classifier. The generator is optimized to minimize the classification loss of the classifier corresponding to the conditioned label. Let C^{MI} denote the classifier

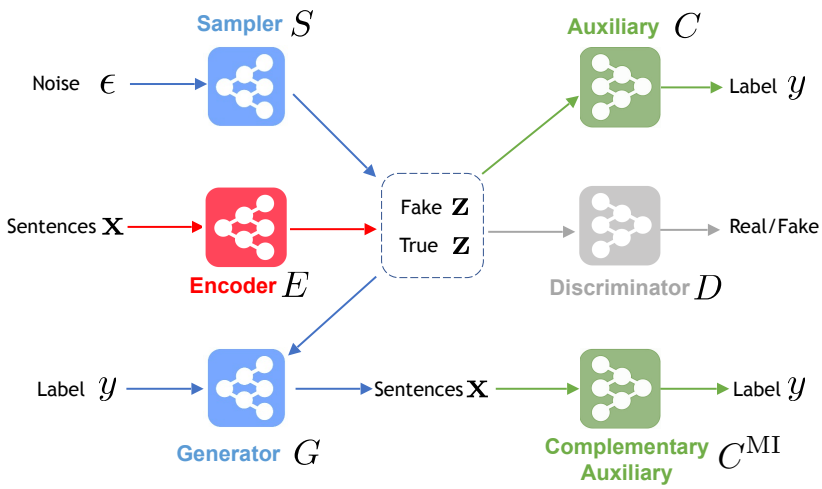


Figure 5.2: Illustration of the proposed CARA method. A complementary auxiliary classifier C^{MI} is proposed to guide the generated sentences to contain the label information. In the training stage, E learns to maximize the classifier loss of C for disentanglement, and G learns to minimize the classifier loss of C^{MI} for conditional generation. In the evaluation stage, only the networks in blue (*i.e.,* the neural sampler S and generator G) are used for label-conditional text generation. Further, we consider BERT as the encoder E, and add C^{MI} with an annealing schedule.

in the output space. The formulation can be written as follows:

$$\max_{E,G,C^{\mathrm{MI}}} \mathcal{L}_{\mathrm{MI}} = \mathbb{E}_{\boldsymbol{z}\sim q_E(\boldsymbol{z}|\boldsymbol{x}),y\sim p(y),\boldsymbol{x}\sim p_G(\boldsymbol{z},y)} \log p_{C^{\mathrm{MI}}}(y|\boldsymbol{x}). \tag{5.7}$$

The full model architecture of CARA is illustrated in Figure 7.1. Since the output of the generator is discrete, this complicates the back-propagation of loss from the classifier in the output space. We therefore follow [124], adopting the Gumbel Softmax distribution for the continuous approximation of discrete output.

Connection to Mutual Information

Note that the generator G and the proposed complementary auxiliary classifier C^{MI} constitute a path for the reconstruction of y, as illustrated in the third row of

Figure 7.1:

$$y \xrightarrow{G} \boldsymbol{x} \xrightarrow{C^{\mathrm{MI}}} y$$

By marginalizing the dependence on $\boldsymbol{z}$ in (5.7), we have the reconstruction loss for y, which is written in its log likelihood form as follows:

$$\mathcal{F}_E = \mathbb{E}_{p\sim p(y),\boldsymbol{x}\sim p(\boldsymbol{x}|y)}[\log p_{C^{\mathrm{MI}}}(y|\boldsymbol{x})]. \tag{5.8}$$

We show that this cycle-consistency term for y maximizes the mutual information between the generated $\boldsymbol{x}$ and its label y. For random variables $\boldsymbol{x}$ and y, the mutual information between $\boldsymbol{x}$ and y can be written as

$$I(\boldsymbol{x}, y) \geq H(y) + \mathcal{F}_E. \tag{5.9}$$

The proof is provided in the next section. Since the label distribution $p(y)$ is known, $H(y)$ is fixed, and $\mathcal{F}_E$ becomes a lower bound for $I(\boldsymbol{x}, y)$. In practice, CARA maximizes $\mathcal{F}_E$ in (5.8), which effectively maximizes the mutual information, enforcing the generated $\boldsymbol{x}$ to contain the label information.

The full objective of CARA is as follows:

$$\min_{E,G} \mathcal{L}_{\mathrm{CARA}} = \mathcal{L}_{\mathrm{ARAE}} + \lambda \mathcal{L}_{\mathrm{MI}}, \tag{5.10}$$

where λ is the weighting hyper-parameter for $\mathcal{L}_{\mathrm{MI}}$. The CARA training procedure is outlined in Algorithm 19.

Note that the proposed CARA model has two classifiers, C and C^{MI}, for predicting labels. Classifier C operates in the latent space, and the encoder is trained to maximize the classification loss so that disentangled features can be learned. The

Algorithm 3: CARA Training

```
for each training iteration do
    (1) Train encoder/ generator for reconstruction
    Sample {x_i, y_i}_{i=1}^n ~ p(x, y), compute z̃_i = E(x_i)
    Minimize E, G wrt L_rec in (5.3)
    (2) Train latent classifier and discriminator
    Sample {x_i, y_i}_{i=1}^n ~ p(x, y), compute z̃_i = E(x_i)
    Sample {s_i}_{i=1}^n ~ N(0, I), compute ẑ_i = S(s_i)
    Minimize C, D wrt L_disentalgle in (5.4) and L_adversarial in (??)
    (3) Train encoder and sampler adversarially
    Sample {x_i, y_i}_{i=1}^n ~ p(x, y), compute z̃_i = E(x_i)
    Sample {s_i}_{i=1}^n ~ N(0, I), compute ẑ_i = S(s_i)
    Minimize E, S wrt L_disentangle in (5.4) and L_adversarial in (??)
    (4) Train output space classifier
    Sample {x_i, y_i}_{i=1}^n ~ p(x, y)
    Minimize C wrt L_MI in (5.7)
    (5) Train encoder and generator adversarially
    Sample {x_i}_{i=1}^n ~ p(x), compute z̃_i = E(x_i)
    Sample y_i from a categorical distribution, compute x̂_i = G(z̃_i, y_i)
    Minimize E, G wrt L_MI in (5.7)
```

other classifier C^{MI} is in the observation space, and the generator is trained to minimize the classification loss so that the generated sentences can be controlled. Therefore, the two auxiliary classifiers are complementary: C^{MI} helps reduce the non-identifiability issue that C causes, and C disentangles $\boldsymbol{z}$ and y so that C^{MI} can effectively constitute a cycle-consistency term to maximize mutual information.

Proofs of Mutual Information

Following [152], on the support of $(\boldsymbol{x}, y)$, we denote q as the complementary auxiliary classifier probability measure, and p as the decoder probability measure. Note that the reconstruction loss for y can be written as its negative log likelihood form as:

$$\mathcal{F}_E = -\mathbb{E}_{p\sim p(y), \boldsymbol{z}\sim p(\boldsymbol{x}|y)}[\log q(y|\boldsymbol{x})]. \tag{5.11}$$

For random variables $\boldsymbol{x}$ and y with two different probability measures, $p(\boldsymbol{x}, y)$ and $q(\boldsymbol{x}, y)$, we have

$$
\begin{aligned}
H_p(y|\boldsymbol{x}) &= -\mathbb{E}_{\boldsymbol{z}\sim p(y),\boldsymbol{x}\sim p(\boldsymbol{x}|y)}[\log p(y|\boldsymbol{x})] \\
&= -\mathbb{E}_{y\sim p(y),\boldsymbol{x}\sim p(\boldsymbol{x}|y)}[\log q(y|\boldsymbol{x})] \\
&- \mathbb{E}_{\boldsymbol{z}\sim p(y),x\sim p(\boldsymbol{x}|y)}\big[\log p(y|\boldsymbol{x}) - \log q(y|\boldsymbol{x})\big] \\
&= -\mathbb{E}_{y\sim p(y),\boldsymbol{x}\sim p(\boldsymbol{x}|y)}[\log q(y|\boldsymbol{x})] \\
&- \mathbb{E}_{p(y)}(\mathrm{KL}(p(y|\boldsymbol{x})\|q(y|\boldsymbol{x}))) \\
&\leq -\mathbb{E}_{p\sim p(y),\boldsymbol{z}\sim q(\boldsymbol{x}|y)}[\log p(y|\boldsymbol{x})],
\end{aligned} \tag{5.12}
$$

where $H_p(y|\boldsymbol{x})$ is the conditional entropy.

From lemma 19, we have For random variables $\boldsymbol{x}$ and y with probability measure $p(\boldsymbol{x}, y)$, the mutual information between $\boldsymbol{x}$ and y can be written as

$$
\begin{aligned}
I_p(\boldsymbol{x}, y) &= H_p(y) - H_p(\boldsymbol{x}|y) \\
&\geq H_p(y) + \mathbb{E}_{y\sim p(y),\boldsymbol{x}\sim p(\boldsymbol{x}|y)}[\log q(y|\boldsymbol{z})] \\
&= H_p(y) + \mathcal{F}_E.
\end{aligned} \tag{5.13}
$$

5.4.4 Generation Quality versus Conditional Accuracy

Though CARA can generate more controllable sentences, due to the proposed auxiliary classifier C^{MI}, it introduces an additional issue on how to tune λ to balance sentence quality (*e.g.,* coherence and faithful linguistic meaning) and label-conditional accuracy (*e.g.,* reflecting the category to which it belongs). When λ is small, the label-agnostic information may dominate, as $p_G(\boldsymbol{x}|\boldsymbol{z}, y)$ tends to degenerate to $p_G(\boldsymbol{x}|\boldsymbol{z})$. Example scenarios include text style transfer where only few sentiment-connected words (*e.g.,*

"like", "hate", "good", and "bad") in a sentence can indicate sentiment and or topic transfer where the number of topical words (*e.g.,* "philosophy", "finance", and "sports") in a sentence (*e.g.,* "I just watched a video about how to play tennis this morning.") can be few. The result is that the model can perfectly resemble the training sentences but fails to change the conditional category when given different conditional labels. When λ is large, the label information can dominate, and $p_G(\boldsymbol{x}|\boldsymbol{z}, y)$ tends to degenerate to $p_G(\boldsymbol{x}|y)$. The model learns to output label-related sentences but sacrifices sentence quality (*e.g.,* "I like like like" and "It is bad, bad, bad"). We propose two practical techniques to alleviate this issue.

Annealing Training Schedule

We first consider an annealing training schedule on λ, to gradually incorporate $\mathcal{L}_{\text{MI}}$ into training without greatly changing the learning from $\mathcal{L}_{\text{rec}}$. Formally, λ has the following form:

$$\lambda_t = \begin{cases} f(\frac{t}{T}), & \text{if } \frac{t}{T} \leq R \\ \lambda_{max}, & \text{otherwise} \end{cases} \tag{5.14}$$

where t is the iteration step, T is the total number of iterations, and f is a monotonically increasing function. There are two hyper-parameters:

- R: the proportion of total iterations used for increasing λ to a plateau.
- λ_{max}: the maximum value λ can reach.

f monotonically increases λ from 0 to λ_{max} using R portion of T iterations.

BERT as a Strong Encoder

As an alternative, we consider improving the encoder so that it provides stronger sentence representation $\boldsymbol{z}$. Recently, models trained with large-scale unlabeled text

Table 5.1: Dataset statistics

Dataset	Attribute	Train	Valid	Test
Personality Captioning	Happy	864	30	39
	Angry	868	26	42
	Malicious	872	14	58
Style Captioning	Humorous	6000	300	300
	Romantic	6000	300	300
	Factual	6000	300	300
Yahoo Questions	Science & Math	126K	14K	6000
	Entertainment & Music	126K	14K	6000
	Politics & Government	126K	14K	6000
Yelp	Positive	270K	2000	500
	Negative	180K	2000	500

data have been shown to provide superior generalization capability when fine-tuned for various language-understanding tasks [5, 6, 7]. These models require millions of parameters; however, they are capable of capturing (approximately) universal language knowledge and generalizing well to different domains. We consider BERT [5]—one of the most prevalent of such large pre-trained models—as our encoder to provide more generalizable sentence representations to balance model learning from different losses and make training more stable.

5.5 Experiments and Results

5.5.1 Datasets

Since label-conditional text generation has been less comprehensively studied, we consider a suite of four datasets to extensively study this problem, as summarized in Table 5.1.

- Personality captioning [153] dataset was proposed for engaging image captioning via personality. Each image in the dataset contains captions with one designated

Table 5.2: Personality captioning results.

Model	Conditional Generation				Attribute Transfer		
	ACC↑	BLEU↑	G-score↑	Self-BLEU↓	ACC↑	BLEU↑	G-score↑
Ctrl-Gen	65.49	22.59	38.46	**6.33**	67.61	22.87	39.32
ARAE	66.66	0.00	0.00	99.98	69.87	0.00	0.00
ARAE-NN-Outlines	**71.83**	20.31	38.20	57.76	88.03	20.31	42.28
CARA	70.42	21.27	38.70	70.18	**91.55**	21.61	**44.48**
CARA_{A}	67.61	22.56	39.05	50.99	84.51	19.55	40.65
CARA_{AB}	61.13	**31.65**	**43.99**	11.68	66.20	**29.67**	44.32

Table 5.3: Style captioning results.

Model	Conditional Generation				Attribute Transfer		
	ACC↑	BLEU↑	G-score↑	Self-BLEU↓	ACC↑	BLEU↑	G-score↑
Ctrl-Gen	41.53	16.44	26.12	61.05	42.60	16.73	26.70
ARAE	43.90	58.53	50.69	99.78	47.02	2.58	11.01
ARAE-NN-Outlines	32.80	**88.92**	**54.00**	99.50	36.67	0.66	4.92
CARA	41.98	63.59	51.67	99.45	42.32	1.08	6.76
CARA_{A}	44.40	47.55	45.95	99.27	**45.47**	1.19	7.36
$CARA_{AB}$	**47.80**	33.96	40.29	**51.08**	40.20	**22.60**	**30.14**

personality. We choose captions from 3 distinctive personalities: `happy`, `angry` and `malicious`. We use only captions and their personality labels.

- Style captioning [154] dataset is a style-based image captioning dataset. We consider three different styles in our experiments: `humorous`, `romantic` and `factual`.

- Yahoo dataset [155] is a topic-based question generation dataset. We choose samples from the `Science` & `Math`, `Entertainment` & `Music`, and `Politics` & `Government` categories. We follow [145] to only generate questions.

- Yelp dataset is a sentiment manipulation dataset containing binary sentiment labels. We follow the setup of [142] for data splitting.

5.5.2 Evaluation

We consider three metrics: (1) *BLEU* for sentence quality, (2) *accuracy* for conditional generation capability (accuracy is assessed by an oracle classifier to correctly predict the attributes upon which the generated sentences are conditioned), and (3) *G-score*, which is reported as the geometric mean of accuracy and BLEU [147]. This is the most important metric, as it evaluates overall performance.

We consider two settings: (*i*) *Conditional generation.* $\boldsymbol{z}$ is generated by the neural sampler S and y is uniformly sampled, $(\boldsymbol{z}, y)$ is used for generation. In this setting, the BLEU of each generated sentence is computed by comparing it with all sentences in the test set, as there are no source sentences. We further report Self-BLEU [141] to evaluate the diversity of generated sentences. (*ii*) *Attribute transfer.* $\boldsymbol{z}$ of a sentence is extracted using the encoder E. It is combined with a different label for transfer generation. In this setting, the BLEU of each transferred sentence is computed by comparing it with its source sentence.

5.5.3 Baselines

We compare with three baselines: (1) *Ctrl-Gen* [124]; (2) *ARAE* [145], as well as a version without using separate generators, which we refer to as ARAE-; and (3) *NN-Outlines* [140], which proposes the use of a general purpose encoder for text generation, and which we implement using BERT. For our ablation study, we consider three variants of CARA:

- *CARA*: the basic model;
- $CARA_A$: CARA with annealing training scheme;
- $CARA_{AB}$: CARA with annealing and BERT encoder.

Table 5.4: Hyperparameters settings of CARA and its variants on different datasets.

Model	Param.	Personality	Style	Yahoo	Yelp
CARA	k_1	10	5	1	1
	k_2	20	10	4	4
CARA_A	k_1	10	5	1	1
	k_2	20	10	4	4
	λ_{max}	0.1	0.5	3.0	
	R	0.5	0.7	0.7	
CARA_AB	k_1	20	3	1	1
	k_2	5	4	4	4
	λ_{max}	0.1	0.5	1.0	10.0
	R	0.5	0.7	0.3	0.7

5.5.4 Training Details

For CARA and its variants, we first pre-train model wrt $\mathcal{L}_{\text{ARAE}}$ for k_1 epochs and then train wrt to the full objective $\mathcal{L}_{\text{CARA}}$ for k_2 epochs. The choices of hyperparameters k_1 and k_2 are shown in Table 5.4. For CARA_AB, we append a 1-d convolutional layer to the BERT model. We fix BERT's weights for the first k_1 epochs and fine-tune the last two layers of BERT for the rest epochs. For all datasets, we use maximum sequence length 21,the vocabulary of BERT, WordPiece tokenization, the temperature for Gumbel Softmax as 0.5, and greedy decoding during inference. We randomly split datasets into train/valid/test by a 7:2:1 ratio if not otherwise specified. For all methods, we draw the same number of samples as that in the test set upper bounded by 5,000 for evaluation. We train a separate BERT-Base model for classification as the oracle classifier on each dataset. The oracle classifiers achieve 73.24%, 83.27%, 79.78%, and 98.54% test set accuracy on personality captioning, style captioning, yahoo and yelp datasets, respectively. For baseline models, we train Ctrl-gen and ARAE- for 12 and 25 epochs, respectively, as default by its released code. On small datasets, including personality captioning and style captioning, we extend the number of training epochs until convergence. We conduct the experiments on 4 NVIDIA K80

Table 5.5: Topic-based question generation results on Yahoo dataset.

Model	Conditional Generation				Attribute Transfer		
	ACC↑	BLEU↑	G-score↑	Self-BLEU↓	ACC↑	BLEU↑	G-score↑
Ctrl-Gen	60.47	64.32	62.37	31.59	60.12	64.21	62.13
ARAE	50.05	22.94	33.88	84.24	37.73	17.06	25.37
ARAE-NN-Outlines	46.79	63.06	54.32	**22.93**	49.38	60.23	54.54
CARA	63.67	64.46	64.06	67.30	68.26	34.10	48.25
CARA_{A}	67.55	69.72	68.63	29.33	69.17	68.96	69.06
CARA_{AB}	**73.49**	**71.90**	**72.69**	26.17	**75.61**	**69.97**	**72.74**

GPUs.

5.5.5 Results and Analysis

We report the results for the four datasets in Tables 5.2, 5.3, 5.5, and 5.6, respectively, and aim to answer the following questions.

Conditional Text Generation

The proposed CARA_{AB} consistently achieves the best G-score of all the datasets, except style captioning. This indicates that CARA is a strong competitor for controllable sentence generation. Meanwhile, the lower Self-BLEU scores of CARA shows that the generated sentence samples of CARA are diverse. As for the two large-scale datasets, Yahoo and Yelp, CARA provides consistent improvements. ARAE achieves similar overall performance as ARAE-, while CARA improves ARAE-. This verifies our assumption that ARAE's solution for alleviating the non-identifiability is less effective than CARA, and sacrifices sentence quality due to fewer training samples for each generator.

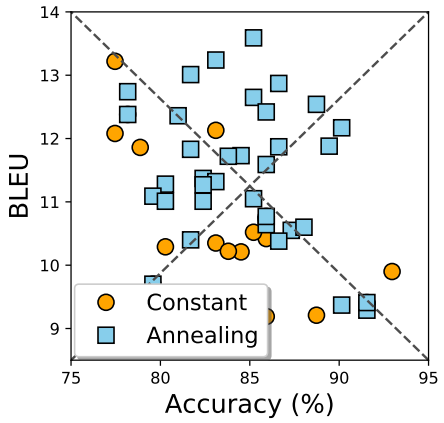


Figure 5.3: Schedule comparison.

Attribute Transfer

Because CARA can be applied to the attribute transfer task, we further compare it with models specifically designed for this task. For the Yelp dataset, we compare with additional previous methods, including Cross-Align [142], MultiDecoder [156], Delete-AndRetrieve [146], StyleTransformer [157], Back-Translation [158], and iVAE$_{\text{MI}}$ [159].

Effectiveness of BERT

The incorporation of BERT yields consistent improvement for both conditional generation and attribute transfer. Note that the intra-class diversity of personality and style captioning datasets are quite high. Most models without BERT show low diversity on the two challenging datasets By incorporating BERT as a strong encoder, the collapse is not observed in CARA$_{\text{AB}}$, demonstrating that a strong universal encoder provides more meaningful latent codes for the generator to rely on in decoding.

Effectiveness of C^{MI} Schedule.

Figure 5.3 compares the constant and annealing schedules for λ. The two schedules with various configurations are performed, and each dot represents the result for each

Table 5.6: Sentiment transfer results on Yelp dataset.

Model	Conditional Generation				Attribute Transfer		
	ACC↑	BLEU↑	G-score↑	Self-BLEU↓	ACC↑	BLEU↑	G-score↑
Cross-Align	-	-	-	-	79.5	12.4	31.40
MultiDecoder	-	-	-	-	47.6	13.25	25.11
DeleteAndRetrieve	-	-	-	-	88.7	14.75	36.17
StyleTransformer	-	-	-	-	93.6	17.1	40.01
iVAE_{MI}	-	-	-	-	92.0	36.7	58.11
Ctrl-Gen	87.81	38.86	58.41	41.23	87.57	37.75	57.50
ARAE	**96.72**	20.18	44.18	25.81	85.43	22.97	44.30
ARAE-NN-Outlines	63.68	34.69	47.00	33.04	61.29	28.62	41.88
CARA	91.45	41.81	61.83	33.16	91.69	43.64	63.26
CARA_{A}	91.45	46.29	65.06	29.74	92.42	46.28	65.40
CARA_{AB}	94.90	**58.88**	**74.75**	**24.95**	**95.45**	**53.25**	**71.29**

configuration. The constant schedule has difficulties balancing BLEU and accuracy, while the annealing schedule can help balance the two metrics, as demonstrated by the large number of dots positioned closer to the diagonal line.

Results of Sentiment Manipulation

Table 5.7 compares CARA with a list of existing methods on sentiment manipulation, including Cross-Align [142], StyleEmbedding [156], MultiDecoder [156], CycleRL [147], Ctrl-Gen [124], ARAE [145], NN-Outlines [140], TemplateBased, RetrieveOnly, DeleteOnly, SimpleCopy, DeleteAndRetrieve [146], FormalityStyle [160], StyleTransformer [157], GlobalVariation [161], UnsuperMT [162], DualRL [163], EditingEntangled [164], Back-Translation [158], and iVAE_{MI} [159].

CARA and CARA_{A}'s overall performance are among the best. CARA_{AB} achieves the best G-score, demonstrating its superior capability for both natural language generation and accurate attribute transfer.

Table 5.7: Sentiment transfer results on Yelp dataset compared with SOTA methods. * indicates generating samples and evaluating using our oracle classifier.

Method	Accuracy	BLEU	G-score
Cross-Align	79.5	12.4	31.40
StyleEmbedding	8.7	19.50	13.02
MultiDecoder	47.6	13.25	25.11
CycleRL	80.00	22.46	42.39
Ctrl-Gen*	87.7	38.86	58.38
ARAE*	85.43	22.97	44.30
NN-Outlines			
TemplateBased	81.7	21.05	41.47
RetrieveOnly	**96.0**	2.9	16.69
DeleteOnly	85.7	13.59	34.13
DeleteAndRetrieve	88.7	14.75	36.17
SimpleCopy	3.00	29.64	9.43
FormalityStyle	82.80	26.44	46.79
StyleTransformer	93.6	17.1	40.01
GlobalVariation	93.78	31.62	54.46
UnsuperMT	95.4	44.5	65.16
DualRL	85.6	**55.2**	68.74
EditingEntangled	95.4	24.6	48.44
Back-Translation	95.4	5.0	21.84
iVAE_{MI}	92.0	36.7	58.11
CARA	91.69	43.64	63.26
CARA_{A}	92.42	46.28	65.40
CARA_{AB}	95.45	53.25	**71.29**
Human	74.0	100	86.0

5.6 Conclusion

We have described the CARA for label-conditional text generation. CARA utilizes one auxiliary classifier for disentangled feature learning in the latent space, and the other auxiliary classifier for more accurate label-conditioning on the generated sentences. Using an annealing training schedule and adopting BERT as a strong encoder further improved CARA's performance. Experiments on four datasets consistently show that CARA achieves both improved natural sentences generation and accurate label transfer.

Chapter 6

Self-Supervised Commonsense Learning

6.1 Introduction

The commonsense learning is a key aspect in learning general knowledge. The commonsense question answering (QA) task aims to examine QA systems' ability to engage in commonsense reasoning in a context that contains complicated logical relationships and implicit universal knowledge. A typical commonsense multiple-choice QA example contains a context, a question, and multiple answer options—among which one and only one is correct—as demonstrated in Figure 6.1 (left). The context provides background support for reasoning and has been proposed in various complex forms, such as procedural descriptions about the change of events and processes [165], people's everyday narratives containing causes and effects [166], and reading comprehension problems in standardized tests comprising logical rules and potential conflicts [167]. In this setting, a commonsense QA solver aims to choose the most plausible correct answer from multiple options based on the context and question. However, it is challenging to solve commonsense QA because questions are intentionally created to examine out-of-context knowledge, and simple similarity matching between questions and the context does not work.

Recently, pre-trained language models, such as BERT [5], RoBERTa [9], and XL-Net [11] have achieved new state-of-the-arts in many natural language processing (NLP) tasks [166, 165, 168]. These models learn generic language representation through self-supervised learning (SSL) on a massive scale of text corpora. Unsurprisingly,

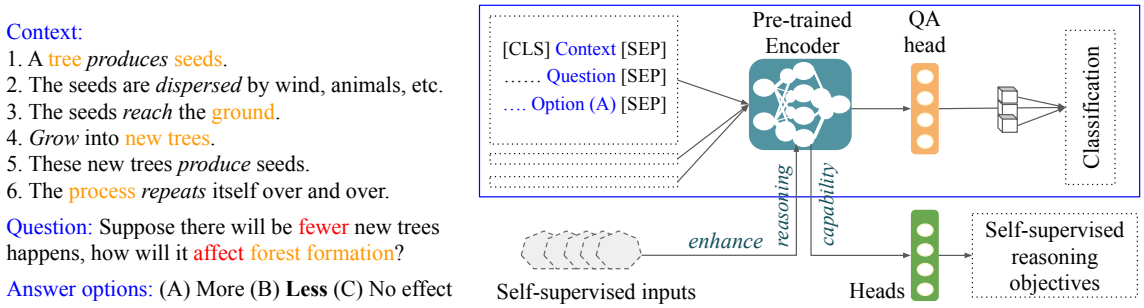


Figure 6.1: Left: An example of commonsense multiple-choice QA. The context contains complex relations among entities (orange text) and dynamically changing processes (italic words). The question asks about the logical influences of changing events or processes. **Right:** Illustration of the proposed elBERto pipeline, where the modules in the blue box form the model backbone, and the rest demonstrate the components for self-supervised commonsense learning. All objectives share the same language encoder to encourage explicitly capturing commonsense knowledge.

on the commonsense QA task, pre-trained models outperform many well-established neural models, even without context or when tested on unseen topics [165]. However, most existing SSL tasks do not explicitly model commonsense knowledge, where the context has intrinsic differences from general text. Thus, a large gap still remains between pre-trained models' performance and human benchmarks. Several attempts have been made to alleviate this issue, such as adapting Transformers to encode extra hops of a textual evidence graph in Transformer-XH [169], injecting knowledge through an attention mechanism [170], and traversing a virtual knowledge base [171]. However, the first attempt suffers from learning implicit rules that may contain data bias, whereas the second and third attempts require a carefully selected knowledge base that contains substantial overlap with the dataset domain and potentially drastically elongated (pretext) training.

To investigate the extent to which SSL can be leveraged for explicit commonsense modeling, we propose a **S**elf-supervised **B**idirectional **E**ncoder **R**epresen**t**ation Learning of C**o**mmonsense (elBERto) [172] pipeline that facilitates explicit commonsense learning and reasoning through Transformer's bidirectional encoder architecture. el-

BERto is equipped with two novel self-supervised tasks and three complementary tasks. First, we propose a novel Contrastive Relation Learning (CRL) task that requires a QA solver to distinguish between a logically valid context and an invalid one with edited but similar content by alternating significant words in the context to their antonyms. Correctly identifying the semantically valid context requires the QA solver to fully understand the local influences between events and commonsense logic. Second, we propose a Jigsaw Puzzle (JP) task to encourage a better understanding of logical chains in long contexts; specifically, the QA solver is presented with multiple sentences extracted from a context and randomly shuffled and optimized toward predicting the correct permutation. Further, we incorporate three SSL tasks proven to be effective in previous works [10, 173, 174, 175] for better sentence and entity understanding, including binary sequence order prediction (BSOP), masked language modeling (MLM), and masked entity modeling (MEM). elBERto has three advantages, which are delineated as follows:

- It utilizes the in-domain training corpus in a data augmentation manner without external data, providing more training signals and achieving data efficiency.
- It does not require extra pretext training time compared to existing methods.
- It reduces the domain gap between seen and unseen context [176, 177] by learning universal commonsense, improving transferability and interpretability.

We conduct comprehensive experiments on elBERto in terms of the three following challenging commonsense QA tasks: WIQA [165], CosmosQA [166] and ReClor [167]. Qualitative and quantitative experiments demonstrate that each proposed SSL task leads to performance improvement over baseline, and elBERto effectively fuses the five tasks, achieving the best accuracy in all datasets. Further, CRL and JP bring

substantial improvement, especially on relatively difficult questions such as out-of-paragraph questions of WIQA and hard-level questions of ReClor. Finally, on both the CosmosQA and ReClor leaderboards, elBERto is in the first place at the time of writing, exceeding all other methods, including those utilizing explicit graph reasoning and external knowledge retrieval. The main contributions of this chapter can be summarized as follows:

- We make the first attempt to present five SSL tasks to empower pre-trained language models with a strong capacity to understand rich logics and knowledge for commonsense QA.
- We propose a novel CRL task to help QA solvers to fully understand local influences between events, and a JP task to encourage a better understanding of logical chains and hidden commonsense in long contexts.
- Our simple yet effective pipeline surpasses all existing works in the three most challenging commonsense QA benchmarks, even those with explicit graph reasoning and knowledge retrieval.

6.2 Related Work

6.2.1 Commonsense QA

Commonsense QA [166, 165, 167] requires understanding and reasoning over real-world knowledge. Most previous approaches to commonsense QA fall into the two following categories: (1) retrieving corresponding evidence or paths from an external knowledge base and serving as data augmentation [178, 170, 179] and (2) establishing context-based graphs with nodes as the keywords extracted from context and then

propagating the information between nodes via Graph Neural Networks (GNNs) [67, 180, 181, 182, 183]. In the first category, Asai *et al.* [178] introduced a new graph-based recurrent retrieval approach that learns to retrieve reasoning paths over the Wikipedia graph to answer multi-hop open-domain questions. Moreover, Huang *et al.* [179] proposed a recursive erasure memory network to cope with the quality improvement of evidence. Ma *et al.* [170] provides a systematic analysis of popular knowledge resources and knowledge-integration methods across benchmarks from multiple commonsense datasets. Zahedi *et al.* [184] proposed a hierarchical compare aggregate model for question retrieval. In the second category, Huang *et al.* [183] proposed a discourse-aware graph network that reasons by relying on the discourse structure of the texts. In addition, Tu *et al.* [182] introduced a heterogeneous graph with different types of nodes and edges.

There are also approaches falling into both categories. For example, in Lin *et al.* 's study [185], the system retrieves external evidence and then performs commonsense reasoning via GNNs. In addition, DA+REG [186] is presented to enrich the training set by replacing words with their antonyms and leveraging logical and linguistic knowledge to augment labeled training data. However, these works require time-consuming retrieval of a large amount of evidence, which is prone to containing noise. Further, GNNs only propagate the information between nodes that have associated effects but fail to involve intervention and counterfactual reasoning. We propose a lightweight pipeline that is capable of performing intervention and counterfactual reasoning, achieving better performance than GNNs.

6.2.2 Pre-trained Language Models (LMs)

The aim of pre-training LMs is to learn universal language representations via web-scale unlabeled data and pretext tasks. Recently, significant pre-trained models, such as BERT [5], RoBERTo [9], XLNet [11], ALBERT [10] and ERNIE [32] have been shown to be effective for downstream NLP tasks. BERT [5] randomly masks out some of the words and trains the model to predict the masked tokens and whether the next sentence will be coherent. In Lan *et al.* [10]'s work, two parameter-reduction techniques are presented to lower memory consumption and increase the training speed of BERT. Li *et al.* [32] propose a word-aligned attention to exploit explicit word information. However, this is not trained specifically for commonsense learning and reasoning over long, complicated contexts. Some recent attempts to adapt pre-trained LMs to commonsense QA (*e.g.*, Zhang *et al.* [174] and Xiong *et al.* [187]) propose to incorporate entity-level knowledge, while Sun *et al.* [188] utilize knowledge graphs. Lan *et al.* [10] used a self-supervised loss that focuses on modeling inter-sentence coherence. However, these approaches require substantial training data and extensive pretext training time. Further, they rely heavily on external knowledge bases. In contrast, elBERto requires little training overhead by leveraging the existing training corpus.

6.2.3 Learning with Auxiliary Tasks

Self-supervised auxiliary tasks have been widely adopted in the field of machine learning. Zhu *et al.* [189] introduce an auxiliary reasoning navigation framework with four self-supervised auxiliary reasoning tasks to take advantage of the additional training signals derived from the semantic information. In reinforcement learning, self-supervised auxiliary tasks are applied to improve data efficiency and robustness [190,

191, 192]. Ma *et al.* [191] introduces a self-monitoring agent with a visual-textual co-grounding module to locate the instruction completed in the past and a progress monitor to ensure the grounded instruction. Veeriah *et al.* [190] presented a novel method for a reinforcement learning agent to discover questions formulated as general value functions. In the NLP domain, SSL has been widely adopted as auxiliary tasks to pre-train a language model [5, 10]. Logeswaran *et al.* [193] present a model that learns the order of sentences in a paragraph, while Jernite *et al.* [173] design three discourse-based objectives over unlabeled text to train a sentence encoder. However, all these methods have problems going beyond representation learning to comprehensively understand entities' relationship, events' causal influence, and the ordering of all context sentences.

6.3 Problem Formulation

In this section, we introduce our setting of a commonsense multiple-choice QA task. In commonsense multiple-choice QA tasks, an input usually contains a context, a question and multiple answer options, among which one and only one is correct, as demonstrated in Figure 6.1 (left). We formulate the input context as $\mathbf{c} = \{c_1, c_2, \ldots, c_n\}$, the question as $\mathbf{q} = \{q_1, q_2, \ldots, q_m\}$, and a candidate answer as $\mathbf{a} = \{a_1, a_2, \ldots, a_k\}$, where c_i, q_i, a_i represent the i_{th} word of the context, question, and answer of lengths n, m, and k, respectively. A QA system aims to choose the most plausible correct answer out of multiple answer options based on the context and the question. The objective can be written as

$$\mathcal{L}_{QA} = -\frac{1}{N}\sum_{i=1}^{N} \log P(\mathbf{a}_i^* | \mathbf{c}_i, \mathbf{q}_i, \{\mathbf{a}_i^t\}_{t=1}^{T}\}), \tag{6.1}$$

where N is the number of training examples, T is the number of answer options, and $\mathbf{a}_i^*$ represents the ground truth answer of the i_{th} example.

6.4 Proposed Method

Figure 6.1 (blue box at the upper right) illustrates our model framework. For each candidate answer $\mathbf{a}_i^t$, the sequences of context, question, and $\mathbf{a}_i^t$ are concatenated via a separator `[SEP]`. The whole sequence is prepended with a beginning-of-sequence token `[CLS]` and appended with a end-of-sequence token `[SEP]`. An encoder module transforms each token in the input sequence to the summation of its corresponding word, position and type embedding, and then it encodes to a hidden representation. The hidden representation of the first token `[CLS]` is then extracted to represent the semantic meaning of the whole sequence and further projected to a single logit via a linear module. The logits obtained by formatting and feeding all candidate answers into the model pipeline are then fed to a softmax layer and optimized by cross-entropy loss. To complement the points that most pre-trained LMs are not explicitly developed for commonsense learning and that little training overhead is preferred, we propose leveraging five self-supervised tasks that approach the problem from distinctive angles. We explain each task below.

6.4.1 Contrastive Relation Learning (CRL)

Commonsense QA contexts usually contain substantial logical influences between events. For example, in the context of Figure 6.1a, each sentence describes an event that is a logical consequence of the event described in the previous sentence (*e.g.,* "the seeds grow into new trees" is caused by "the seeds reach the ground"). Fully understanding such a context requires recognizing the causes and consequences between different events. Moreover, it is observed that when changing one or multiple words in a context to their opposite meaning, the semantic meaning of each sentence may still be valid, but the local influences between sentences (events) are usually lost.

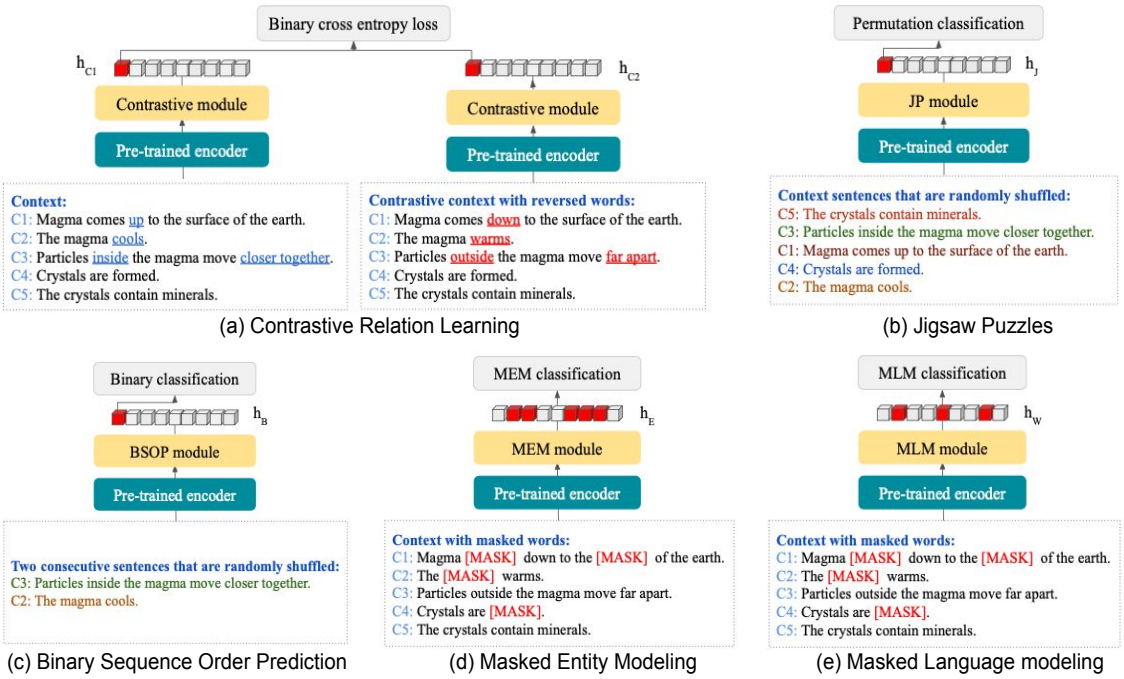


Figure 6.2: Demonstration of the proposed Contrastive Relation Learning (CRL), Jigsaw Puzzle (JP), Binary Sequence Order Prediction (BSOP), Masked Entity Modeling (MEM), and Masked Language Modeling (MLM) SSL tasks. All tasks share the same pre-trained encoder to encourage better capturing of commonsense knowledge.

For example, when replacing "reach" with its antonym "move away," the resulting event—"the seeds move away from the ground"—is semantically reasonable, but it will not result in "the seeds grow into new trees." While it is easy for humans to identify such implicit logical errors (seeds cannot grow without soil or nutrition), this is difficult for machines because there is no commonsense knowledge to refer to. To improve a QA solver's capacity to recognize commonsense logic, we design a CRL task by flipping significant words in a context to their antonyms; this generates contrastive context involving counterfactual logic. Let $\mathbf{c}_i$ and $\mathbf{c}'_i$ denote the original and corresponding contrastive contexts, respectively. We train the QA solver to identify the valid context from $(\mathbf{c}_i, \mathbf{c}'_i)$, which appear in random order; in this way, commonsense knowledge is learned. Further, we create a list of frequently occurring words associated with causes and effects for easy generation of pairs of contrastive

contexts. Mathematically, the objective of CRL can be written as

$$\mathcal{L}_{CRL} = -\frac{1}{N}\sum_{i=1}^{N} \log P(\mathbf{c}_i|(\mathbf{c}_i, \mathbf{c}'_i)). \tag{6.2}$$

As illustrated in Figure 6.2a, the CRL task is implemented by appending a contrastive module to the pre-trained encoder for converting the encoded representation of each of a pair of contrastive contexts into one-dimensional logits. The logits are then fed to cross entropy loss for optimization. The encoder is shared between the CRL objective (Eq 6.2) and the QA objective (Eq 6.1) to empower the encoder with improved commonsense learning capability.

CRL utilizes a contrastive approach (*e.g.*, contrastive loss) that is close to contrastive learning of visual representation [18, 2], contrastive learning of graph embeddings [194, 195, 196, 197] and structured environment abstraction [198, 199]. However, elBERto differs from them in both the data formation and end goals. Previous approaches have constructed augmented samples and learned to identify these matched pairs from the other of contrastive samples to force the model to be invariant to data augmentation. elBERto constructs contrastive samples and trains the model to recognize the only logical sound one between them, aiming to force the model to capture commonsense logical rules. In addition, while Oscar [200] uses contrastive loss—similar to elBERto—to differentiate image representations from polluted representations, this model was initially proposed for object-semantics pre-training. Finally, CRL possesses a prominent design merit for commonsense QA by forming contrastive contexts with reversed words because questions in QA tasks often ask about the change of effects by alternating words describing quantity or severity level. For example, the question in Figure 6.1a asks about the effects of "fewer" new trees, which contrasts with the event of "more" trees described in the context (seeds "grow into new trees"). Thus, training a QA solver with the CRL objective facilitates understanding

logical influences between events and the acquisition of commonsense knowledge that generalizes to unseen scenarios.

6.4.2 Jigsaw Puzzles (JPs)

Besides understanding the validity of direct logical influences between events through CRL, it is also important to understand indirect influences scattered across long contexts. For example, "a tree produces seeds" can lead to indirect results of "more forest formation." However, without the capability of inferring the missing intermediate logical chains or reasoning over the provided context for the correct chain, the model cannot come to this conclusion. However, the longer the influence chain is, the harder it is to trace the events and reasoning over effects [165]. To help alleviate this problem, we propose a JP task where segments in a paragraph are randomly permuted, and the model is trained to choose the correct order out of K presented permutations. The segments are either one sentence or multiple contiguous short sentences. Correctly restoring the sentence order requires the model to understand not only the semantic meaning of each individual sentence but also the long-chain influences along with them, and thus, it must capture indirect implications in different lengths of sentences. Let $(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m)$ represent a sequence of consecutive sentences and $(\mathbf{s}_1^k, \mathbf{s}_2^k, \ldots, \mathbf{s}_m^k)$ represent the k_{th} permutation of the sequence, where $1 \leq k \leq K$. The objective of JP can be written as

$$\mathcal{L}_{JP} = -\frac{1}{N} \sum_{i=1}^{N} \log P(r|\{(\mathbf{s}_1^k, \mathbf{s}_2^k, \ldots, \mathbf{s}_m^k)\}_{k=1}^{K}), \tag{6.3}$$

where r is the index of the correct order. To incorporate the JP objective, as illustrated in Figure 6.2b, we append a JP module to the pre-trained encoder such that encoded features of each context with shuffled sentence order are fed to the module and converted to one-dimensional logits. The logits are then optimized via cross-entropy

loss.

6.4.3 Additional Tasks

We incorporate three classic SSL tasks to facilitate understanding of causal relationships, entity relationships, and language semantics.

Binary Sentence Order Prediction (BSOP)

Predicting sentence order has been proposed for learning better sentence representation [10] and discourse-level coherence [173, 201, 202]. Based on similar techniques, we adapt a BSOP task to commonsense QA for a better understanding of causal relationships. Since consecutive sentences in QA examples usually exhibit strong causal relationships, training a QA solver to identify the correct order between two continuous sentences can lead to a better understanding of context. To achieve this, we sample subsequent sentences $(\mathbf{s}_i, \mathbf{s}_{i+1})$ from the training corpus and randomly shuffle their order, as demonstrated in Figure 6.2c. Then, we train the model to predict the correct order of each pair of shuffled sentences $(\mathbf{s}'_i, \mathbf{s}'_{i+1})$. The objective of BSOP can be written as

$$\mathcal{L}_{BSOP} = -\frac{1}{N}\sum_{i=1}^{N} \log P(o|\{(\mathbf{s}'_i, \mathbf{s}'_{i+1})\}_1^2), \tag{6.4}$$

where o represents the correct order between $\mathbf{s}'_i$ and $\mathbf{s}'_{i+1}$. As illustrated in Figure 6.2c, we attach a BSOP module to the encoder and follow the same process as in CRL and JP to train the BSOP task. It is worth noting that BSOP is not a JP subtask, although the two may exhibit the same examples when K=2. In JP, there are always fixed K (*e.g.*, 5) sentences. If a context contains more (less) sentences, short (long) ones are merged (split). However, BSOP considers only two consecutive sentences (among all of them) from the contexts. Given this setting, BSOP focuses on modeling direct

relations between consecutive sentences, whereas JP focuses on modeling long-term indirect influences scattered across sentences.

Masked Entity Modeling (MEM)

To encourage better entity relationship encoding, we propose incorporating an MEM [174, 175, 203] task to learn entity relationships embedded in a context. Let $\mathbf{s}$ denote a sequence example, where $\mathbf{s}_E$ and $\mathbf{s}_{\setminus E}$ denote the masked entities and remaining unmasked tokens, respectively. The objective of MEM can be written as

$$\mathcal{L}_{MEM} = -\frac{1}{N}\sum_{i=1}^{N} \log P(\mathbf{s}_{\mathcal{E}}|\mathbf{s}_{\setminus\mathcal{E}}). \tag{6.5}$$

As shown in Figure 6.2d, encoded features of the `[CLS]` token generated by the encoder are fed to an MEM module to predict entities, and they are further optimized by cross-entropy loss on the masked entity tokens.

Masked Language Modeling (MLM)

We also incorporate MLM loss [10] to enhance the model's capability to understand language semantics. This task works well, especially for a scenario where context and questions consist of synonyms, such as "table" and "desk," "improves" and "increases," "lead to" and "results in." Following common practice in MLM [5], we randomly choose 15% of the words in a context sequence and replace them with `[MASK]`. Let $\mathbf{s}$ denote a sequence example, where $\mathbf{s}_M$ and $\mathbf{s}_{\setminus M}$ denote the masked tokens and the remaining unmasked tokens, respectively. The objective of MLM can be written as

$$\mathcal{L}_{MLM} = -\frac{1}{N}\sum_{i=1}^{N} \log P(\mathbf{s}_M|\mathbf{s}_{\setminus M}). \tag{6.6}$$

As shown in Figure 6.2e, encoded features of the `[CLS]` token generated by the encoder are fed to an MLM module to predict tokens, then optimized by cross-entropy loss on the masked tokens.

6.4.4 Overall Objective

Finally, the objectives of the QA task and the five presented SSL tasks are combined to optimize the commonsense QA solver. The final objective is written as

$$\mathcal{L} = \mathcal{L}_{QA} + \alpha\mathcal{L}_{CRL} + \beta\mathcal{L}_{JP} + \gamma\mathcal{L}_{BSOP} + \lambda\mathcal{L}_{MEM} + \delta\mathcal{L}_{MLM}, \tag{6.7}$$

where $\alpha, \beta, \gamma, \lambda, and \delta$ control the importance weight of each SSL objective.

6.5 Experiments and Results

We evaluate elBERto's effectiveness on commonsense QA datasets and perform an ablation study on individual SSL tasks and their performance for different question types. Specifically, we examine whether each of the proposed SSL tasks is effective for learning commonsense and whether SSL is able to fuse the five SSL tasks robustly for comprehensive improvement over baseline and individual SSL tasks; this is because an increased number of tasks also increases computational complexity, which may alternatively compromise accuracy.

6.5.1 Datasets

We evaluate the proposed method on three commonsense QA datasets—namely, WIQA [165], CosmosQA [166], and ReClor [167]. Table 7.1 shows the statistics for each dataset. The three datasets can be described as follows:

- **WIQA** [165] is a recently proposed dataset asking "what-if" questions. It contains 29,808, 6,894, and 3,003 training, validation, and testing samples, respectively. Depending on whether the supporting knowledge resides in the given text, the questions are categorized into "in-paragraph," "out-of-paragraph,"

Table 6.1: Dataset statistics. In/out/no represent in-paragraph/out-of-paragraph/no-effect question types.

Dataset	Total	Train	Validation	Test	Question Types	Question Type Split
WIQA	39705	29808	6894	3003	in/out/no	9893/17108/13694
CosmosQA	35210	25262	2985	6963	-	-
ReClor	6138	4638	500	1000	easy/hard	440/660

and "no-effect" types. In-paragraph and out-of-paragraph question types refer to examples in which the inquired events or entities can and cannot be found in the context, respectively. No-effect-type questions ask about events or entities not related to the context. Out-of-paragraph questions are more difficult to solve than in-paragraph ones are because external commonsense knowledge is required.

- **CosmosQA** [166] focuses on human daily life scenarios. It consists of 35,210 samples, which are divided into 25,262, 2,985, and 6,963 training, validation, and testing samples, respectively.

- **ReClor** [167] is collected from standardized graduate admission examinations. It contains 6,138 samples, which are divided into 4,638, 500, and 1,000 training, validation, and testing samples, respectively. According to biases contained in the data points, the test set is separated into EASY (Test-E) and HARD (Test-H) sets.

6.5.2 Baselines

On WIQA, we compare with Majority [165], Polarity [165], Adaboost [204], Decomp-Att [165], and BERT (base/large) finetune [5]. On CosmosQA, we compare with both base-scale models, including Commonsense-Rc [205], GPT-FT [6], BERT-base [5], and KagNet [185], and large-scale models, including DMCN [206], [5], BERT-large

Multiway [166], RoBERTa-large [9], and ALBERT-xxlarge [10]. On ReClor, we compare with FastText [207], Bi-LISTM [208], GPT (1/2) [6], BERT (base/large) [5], XLNet (base/large) [11], and RoBERTa (base/large) [9].

6.5.3 Implementation Details

Model Implementation and Optimization

We use BERT [5], RoBERTa [9], and ALBERT [10] as the encoders, with a dropout probability of 0.1. Input sequences are limited to 180 tokens. The learning rates for the BERT-base/RoBERTa-base and BERT-large/RoBERTa-large encoders are 1e-5 and 1e-6, respectively. The warm-up step is set to 10% of the total training steps. We train 10 epochs with batch size 4 and Adam optimizer; $\alpha, \beta, \gamma, \lambda, and \delta$ in Eq 6.7 are all equal to 0.2.

Self-Supervised Task Details

For the MLM task, we follow BERT to mask input tokens at random with a probability of 15%. If the i_{th} token is chosen, we replace the i_{th} token with [MASK] token 80% of the time, a random token 10% of the time and the original i_{th} token 10% of the time. For the MEM task, we use TAGME to extract entities because it contains more entities vocabulary compared with the NER tagger, and it has been widely adopted [178, 169, 209]. We then mask the entities with 15% probability. For the JP task, because samples have different numbers of sentences, we process them such that all samples contain exactly K sentences. For any sample containing sentences not equal to K, the following applies: (i) if the total number of sentences in a sample is smaller than K, the longest sentence is decomposed according to punctuation or conjunction words; or (ii) if the total number of sentences is greater than K, the

Table 6.2: Test accuracy (%) on WIQA data. In, out, no represents in-paragraph, out-of-paragraph, no-effect, respectively.

Methods	In	Out	No	Total
Majority	45.46	49.47	0.55	30.66
Polarity	76.31	53.59	0.27	39.43
Adaboost	49.41	36.61	48.42	43.93
Decomp-Att	56.31	48.56	73.42	59.48
BERT-base	70.57	58.54	91.08	74.26
elBERto (BERT-base)	**72.83**	**63.86**	**92.11**	**77.22**
BERT-large	74.91	67.08	**90.20**	78.12
elBERto (BERT-large)	**77.74**	**73.48**	87.73	**80.19**
RoBERTa-base	76.60	64.78	**89.48**	77.19
elBERTo (RoBERTa-base)	**78.49**	**74.14**	86.29	**79.99**
RoBERTa-large	76.79	66.42	**91.79**	78.85
elBERTo (RoBERTa-large)	**79.06**	**74.47**	88.69	**81.22**

two shortest sentences are merged. We repeat (*i*) and (*ii*) until the final number of sentences is K, which is set to 5 in our experiments. For BSOP, we randomly select adjacent pairs of sentences and concatenate them with corresponding context sentences individually. Then, we reverse their order with 50% probability. Sentence pairs in the reversed order are set as negative samples, and sentence pairs in the original order are set as positive samples. For the CRL task, we develop a list of qualitative words and their corresponding antonyms. [1]

6.5.4 Results and Analysis

Comparison with State-of-the-Art Methods

First, we verify whether elBERto can effectively exploit additional training signals from existing data and only an external antonym list. As [166] and our experiments

[1]The antonyms are obtained from `https://www.enchantedlearning.com/wordlist/opposites.shtml` and `https://eslforums.com/antonyms/`, and they contain 658 antonyms. A full list can be downloaded from `https://github.com/anonymity2/AntonymsList`.

Table 6.3: Accuracy on CosmosQA data.

Model	Dev	Test
Sliding Window	25.0	24.9
Standford Attentive Reader	45.3	44.4
Gated-Attention Reader	46.9	46.2
Co-Matching	45.9	44.7
Commonsense-Rc	47.6	48.2
GPT-FT	54.0	54.4
BERT-base	60.4	62.9
KagNet (BERT-base)	-	64.9
elBERto (BERT-base)	**65.3**	**66.3**
BERT-large	66.2	67.1
DMCN (BERT-large)	67.1	67.6
BERT-large Multiway	68.3	68.4
elBERTo (BERT-large)	**69.7**	**69.6**
RoBERTo-base	69.7	71.0
elBERTo (RoBERTo-base)	**72.0**	**73.6**
RoBERTa-large	81.2	81.7
elBERto (RoBERTa-large)	**82.6**	**82.9**
ALBERT-xxlarge	-	85.4
elBERto (ALBERT-xxlarge)	**86.3**	**86.1**
Human	-	94.0

show no significant improvement is achieved when further increasing training samples after 15K questions; it is not easy to learn commonsense knowledge by simply stacking more training samples. Tables 6.2, 6.3, and 6.4 demonstrate the experimental results, from which we draw several observations:

- On all three datasets, elBERto achieves the new state-of-the-art results compared to all baseline methods, demonstrating that it successfully learns better commonsense knowledge through limited data.

- When applied to 12-layer pre-trained models, elBERto achieves a larger improvement than being applied to 24-layer pre-trained models, indicating that it

Table 6.4: Accuracy (%) on the ReClor dataset. Test-E and Test-H represent easy and hard level test questiongs.

Model	Val	Test	Test-E	Test-H
FastText	25.0	25.0	25.0	25.0
Bi-LISTM	27.8	27.0	26.4	27.5
GPT	47.6	45.4	73.0	23.8
GPT-2	52.6	47.2	73.0	27.0
XLNet-base	55.8	50.4	75.2	30.9
XLNet-large	62.0	56.0	75.7	40.5
BERT-base	54.6	47.3	71.6	28.2
elBERTo (BERT-base)	55.4	49.1	71.4	31.6
BERT-large	53.8	49.8	72.0	32.3
elBERTo (BERT-large)	58.4	51.8	76.6	32.3
RoBERTa-base	55.0	48.5	71.1	30.7
elBERto (RoBERTa-base)	**58.4**	**52.0**	**76.6**	**32.7**
RoBERTa-large	62.6	55.6	75.5	40.0
elBERto (RoBERTa-large)	**65.0**	**57.3**	**75.7**	**42.9**
Human (Graduate Students)	-	63.0	57.1	67.2

brings more benefits when a less capable encoder is used; this provides potential advantages for low-resource training.

- Moreover, on both CosmosQA and ReClor leaderboards, elBERto is in first place, exceeding all methods—including those utilizing explicit graph reasoning and external knowledge retrieval, such as KagNet [185] (elBERto outperforms it by 1.4%).

Effectiveness Verification of Each Self-Supervised Task

We study the performance of each proposed task on commonsense QA. The results are demonstrated in Table 6.5, from which we draw several observations:

- Each task boosts performance over baseline on all datasets, demonstrating their

Table 6.5: Comparison of individual self-supervised tasks, elBERto and baseline.

#	MLM	MEM	BSOP	JP	CRL	WIQA	CosmosQA	ReClor
1						74.26	60.54	55.00
2	✓					75.39	62.98	55.60
3		✓				76.49	62.98	56.00
4			✓			75.66	64.42	56.80
5				✓		76.39	63.42	57.00
6					✓	76.76	63.85	57.60
7				✓	✓	77.09	64.09	58.00
8	✓	✓	✓	✓	✓	**77.22**	**65.26**	**58.40**

effectiveness.

- The combination of the novel CRL and JP tasks outperforms each individual task on both WIQA and ReClor datasets, and are only 0.33% lower than BSOP. This shows that joint training of CRL and JP leads to greater benefits for learning commonsense reasoning.

- elBERto achieves the best results by combining all five tasks.

On the more difficult CosmosQA and ReClor datasets, elBERto leads to the most improvement with 4.82% and 3.40% increased accuracy, respectively. (*iv*) Even though WIQA might be biased toward the CRL task because its question formation process is similar to CRL's example construction, CRL achieves 3.41% and 2.60% accuracy boosts over the best baseline on CosmosQA and ReClor, which have no bias, demonstrating that it indeed effectively learns commonsense knowledge.

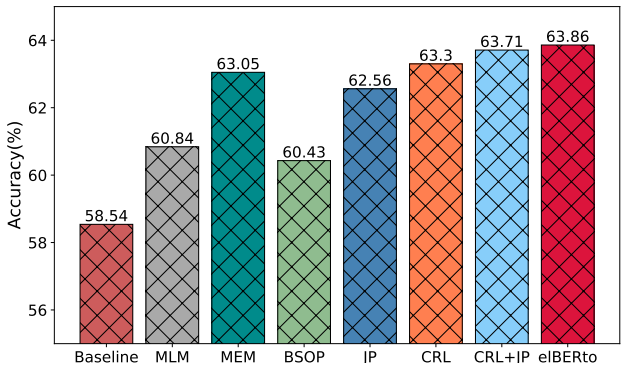


Figure 6.3: Results on WIQA Out-of-Paragraph questions.

Evaluation of Out-of-Context and Hard Questions

We examine elBERto's ability to address hard questions, including those where the answers are not shown in the given context and those containing complex implicit logical rules. Specifically, on the most difficult out-of-paragraph question type of WIQA, where the answers are out of context and cannot be solved by simple similarity comparison between answers and contexts, elBERto obtains 5.32% and 6.40% performance boosts over BERT-base and BERT-large fine-tuning, respectively, as shown in Table 6.2 (third column). This verifies that elBERto effectively learns out-of-context commonsense and can use it accurately. Further, Figure 6.3 shows the performance of each task in elBERto, in which CRL, JP, and MEM boost performance over the baseline by 3.41%, 2.98% and 2.53%, respectively. In addition, on hard-level test samples in the ReClor dataset (Table 6.4 (last column)), elBERto achieves 2.0% and 2.9% accuracy improvements over RoBERTa-base and RoBERTa-large, respectively. Despite using the same pre-trained encoder, elBERto demonstrates superior understanding capability over complex contexts.

	WIQA	WIQA	CosmosQA
Context	**[1]** Bees **land on** (take off) flowers and eat the nectar of the plants. **[2]** While on the flowers pollen **gets on** (gets off) the bee's fur. **[3]** The bee flies to another area with other flowers. **[4]** The pollen is **deposited** (withdraw) in that area. **[5]** A plant is **able** (unable) to reproduce.	**[1]** The moon orbits the earth. **[2]** The earth orbits the sun. **[3]** The moon comes between the sun and the earth. **[4]** The sun is totally **blocked** (unblock). **[5]** A solar eclipse is **occurring** (disappear).	**[1]** We had a sales meeting today and one of our team, the soon to be married Tim, **[2]** said we were out of stress balls - though I am not sure of any connection to Tim 's impending nuptials. **[3]** I figured the **best** (worst) way to show we are not out of stress balls was to dump a box of 300 balls over him. **[4]** We all had a **laugh** (cry) and talked about the value of the stress balls at trade shows and in demonstrations. **[5]** We have used these balls at four trade shows now and people **love** (hate) them .
Question	Suppose the weather is good for bees happens, how will it affect less flowers being pollinated? (a) More **(b)** Less (c) No effect	Suppose faster orbits of the moon happens, how will it affect less blockage of earth? **(a)** More (b) Less (c) No effect	What makes these stress balls so valuable ? **(a)** They can give them away for free at trade shows and its a form of marketing. (b) None of the above choices. (c)They hold no value , I was being sarcastic. They sell them at trade shows to get a little more profit
Prediction	BL, MLM, BSOP: (a) MEM: (c) **JP, CRL, JP+CRL, elBERto**: **(b)**	BL, MLM, MEM: (b) BSOP: (c) **JP, CRL, JP+CRL, elBERto:** **(a)**	BL: (d) MLM, MEM: (b) BSOP: (c) **JP, CRL, JP+CRL, elBERto**: **(a)**

Figure 6.4: Prediction examples of elBERto, individual SSL tasks, and baseline. Yellow highlighted words are masked entities in MEM. Words in red are used for CRL. Grey-shadowed sentences are used for BSOP. The gold answers are in blue.

Evaluation of Out-of-Domain Datasets

In this section, we examine the transferability of elBERto to out-of-domain datasets. Specifically, we train elBERto and BERT on the WIQA dataset comprising scientific procedures and test them on CosmosQA, which contains daily live human conversations, forming distinct source and target domains. elBERto, BERT, and random guessing achieve 31.90%, 27.87%, and 25% accuracy, respectively, demonstrating that elBERto has better transferability to unseen domains.

Qualitative Results

Figure 6.4 demonstrates the visualization results. It can be observed that the baseline fails to predict the correct answer for all examples. JP, CRL, JP+CRL, and elBERto succeed in answering them all correctly, demonstrating their effectiveness for commonsense reasoning in dynamic contexts.

Context	Question & Answer Options
[1] DNA replication is initiated by proteins, **[2]** The hydrogen bonds that hold together a strand of DNA are broken, **[3]** A polymerase reads each side of the strand, **[4]** Adds complimentary nucleotides, Once all the bases are matched, the new DNA strand is formed, **[5]** DNA ligase seals the new pieces into two new strands.	Suppose less hydrogen available happens, how will it affect hurting the DNA to replicate properly. **(a)** More **(b)** Less **(c)** No effect
[1] A tadpole is hatched, The tadpole eventually grows hind legs, **[2]** The tadpole loses its tail,The tadpole becomes a frog, **[3]** The frog mates with another frog to produce more offspring, **[4]** A tadpole is hatched, **[5]** The cycle begins again.	Suppose oil tanks near the pond spill happens, how will it affect less frogs. **(a)** More **(b)** Less **(c)** No effect
[1] Blood delivers oxygen in the body, Proteins and acids are broken down in the liver, **[2]** The liver releases waste in the form of urea, **[3]** The blood carries the urea and carbon dioxide to the kidneys, **[4]** The kidneys strain the urea and salts needed from the blood, **[5]** The carbon dioxide by product is transported back to the lungs, Carbon dioxide is exhaled by the lungs.	Suppose during juicing happens, how will it affect more exhaling. **(a)** More **(b)** Less **(c)** No effect

(a)

Context	Question & Answer Options
[1] I spoke with mom a few days before Baby E 's 6month session . **[2]** She was worried because he was n't sitting up yet that his photos would be the same as his 3month ones . **[3]** I assured her that he would have WAY more personality in these and we could do some fun things with him . **[4]** I brought along props to sit him in to help him sit up and we went to town shooting .... **[5]** this turned out to be one of my favorite sessions I have EVER had .	Is it important to Baby E to have his photo taken ? **(a)** Baby E is excited to have is his photo taken no matter the time . **(b)** Baby E was annoyed with the quality of the 3 month photos and wanted to redo them . **(c)** Baby E has a personality that likes to go to town shooting photos . **(d)** None of the above choices .
[1] Another thing I do n't appreciate is the shooting games . **[2]** Jon had bought this shooting game and one day I watched the little movie intro to the level he was about to play . **[3]** It was imitating World War 2, **[4]** and Jon was on the team that was going and killing the Germans . **[5]** I had big issues with this .	Why might I have problems with Jon playing a shooting game ? **(a)** Because a shooting game involves violence but do n't kill people . **(b)** Because a shooting game involves fun and hitting people . **(c)** Because a shooting game involves violence and killing people . **(d)** Because a shooting game involves fun and killing people .

(b)

Figure 6.5: Examples of failing cases of elBERto on the WIQA (a) and CosmosQA (b) datasets. Green text is the ground truth answer. Red text shows the predictions of elBERto.

Failing Case Analysis

We observe that the performance of the out-of-paragraph question type is lower than the performances for the in-paragraph and no-effect types, although great improvement is achieved in this type. Most failing cases are due to insufficient specialized knowledge, such as knowledge about DNA replication, in the context. Figure 6.5 demonstrates failing cases of elBERto on the WIQA and CosmosQA datasets.

6.6 Conclusion

In this chapter, we proposed a novel self-supervised commonsense learning pipeline that better captures commonsense in a complicated multi-hop context and contains

the two following novel tasks: (*i*) CRL of logical influences between events, achieved by flipping words to their contrastive semantics, and (*ii*) a new JP task for inferring logical chains in a long context by randomly shuffling sentences. Three classic SSL tasks were also incorporated into the model to improve language encoding ability. On the three challenging commonsense QA datasets, our simple yet effective method achieved the best performance compared with baseline methods, including those utilizing explicit graph reasoning and external knowledge retrieval

Chapter 7

Visual-Linguistic Masked Autoencoders

7.1 Introduction

All the previous methods proposed in the second part of this dissertation focus on language domain, such as representation learning, text generation, and commonsense QA. Although Transformers are initially proposed for language tasks, increased research interest has been aroused to extend Transformers to the vision domain and visual representation learning. The traditional predominant CNNs succeed in efficiently extracting image features by convolutional operations, which considers the local attention for each pixel and reduces the image sizes greatly to a moderate latent dimension, substantially alleviating the computational burden of the model. However, when adopting a Transformer model for image encoding, without the convolutional operations, all image pixels are encoded by the Transformer, leading to infeasible computational complexity and prevents the wide spread of Transformers in the vision domain. To alleviate this problem, DETR [36] proposes staking a convolutional layer and a sequence of Transformer encoder layers to effectively encode images. The first convolutional layer extracts image features from the high-dimensional input pixels, decreasing the input size to a manageable size. Then, the subsequent Transformer encoder layers fuse the features globally, resulting in effective learning of image representation. To avoid the use of a convolutional layer, Vision Transformers (ViTs) [16] propose leveraging a large Transformer for image classification by first representing an image as a sequence of patches obtained by segmenting and summarizing image

pixels in a non-overlap manner and then feeding the patches into an Transformer encoder for global fusion. The generated features of the ([CLS]) token are then used for image classification. The patching [16] approach is later widely adopted in the computer vision community for image representation learning with Transformers. Recently, CLIP [210] advances vision representation learning through the extra language supervision by leveraging a large corpus of image-text pairs for training a contrastive learning model. This approach can also be interpreted as visual-linguistic representation learning as both vision and language modalities are jointly learned in the same network. Inspired these appalling methods, we follow this line of work to improve vision representation learning via additional language supervision and developing Transformer-based architectures.

Visual-linguistic representation learning is a classic and challenging problem to bridge the vision and language domains via learning of a common semantic space. Early work such as ViLBERT [211], LXMERT [212], Pixel-BERT [213], and InterBERT [214], proposed global attention-based layers to fuse image and text features which are obtained from two separate encoders implemented by the Transformer, convolutional neural networks (CNNs), or other architectures. Inspired by the success of the Transformer in NLP, later work adopted a unified Transformer model to jointly model visual and linguistic information, such as VL-BERT [215], VisualBERT [216], Unicoder-VL [217], UNITER [218], and Oscar [200]. However, most of these methods rely on an auxiliary Fast(er) R-CNN to extract image regions-of-interest (RoIs) features, which are then concatenated with text features before feeding theem to the single Transformer. The Fast(er) R-CNNs are mostly held fixed (not trained) in these frameworks, preventing the model from being end-to-end trainable or achieving peak performance and robustness.

Moreover, the success of BERT [5] has proven the effectiveness of masked language

modeling (MLM) tasks which randomly change tokens in the input to a MASK token or tokens drawn from the dataset, and then encourages the model to recover the masked tokens. Applying similar mechanisms such as predicting missing image content to learn visual representation, which we call Masked Image Modeling (MIM), and learning both visual and linguistic representation have attracted substantial interests in the research community. Most recent work is based on the Transformer, such as iGPT [219], which learns a Transformer decoder to generate the next image pixel based on the previous seen image pixels, and BEiT [220] which leverages an autoencoder to learn a latent space of discrete visual tokens and a BERT-like model to predict visual tokens for masked image patches conditioned on the unmasked patches. Lately, Masked Autoencoders (MAEs) [14] were proposed to facilitate image representation learning by masking image pixels with fairly high masking ratio and reconstruction loss on the missing pixels, demonstrating superior performance when transferred to downstream vision tasks.

Inspired by the similarity between MIM and MLM, it is natural to consider if they can be leveraged jointly for image and language modeling. To achieve this, we propose the **V**isual-**L**inguistic **M**asked **A**uto**E**ncoder (**VL-MAE**), which consists of a unified Transformer encoder-decoder architecture for visual-linguistic modeling. We then employ the proposed Masked Visual-Linguistic Modeling (MVLM) pretext task to integrate MIM and MLM. Specifically, VL-MAE takes a partial observation of image-text pairs as input and reconstructs the missing image patches and text tokens via MVLM. Compared to previous methods that require a separate detection model such as the Faster R-CNNs to extract image RoIs and features, the proposed VL-MAE is end-to-end trainable.

We conduct extensive experiments on the ImageNet dataset including linear probing, fine-tuning, zero-shot classification, and visualization. When transferred

to a zero-shot image classification task, VL-MAE generates accurate class labels for unseen images without any fine-tuning, demonstrating its effectiveness in aligning visual and linguistic information via masked visual-linguistic modeling. We further visualize the word cloud for each ImageNet category by selecting the words in the top five predicted category labels for each image in the category, and plot the word cloud to reflect word frequency. Our results indicate that the proposed model is able to predict accurate categorical words for common categories. For rare categories that do not appear in any textual descriptions in the pre-training dataset, our model is able to find similar words or encompassing (parent) words, indicating that the model has successfully learned the concept of categories.

7.2 Related Work

7.2.1 Masked Language and Image Modeling

The masked language modeling (MLM) pretext task randomly changes tokens in the textual input to MASK tokens or tokens sampled from the dataset, and then restores the masked tokens. It was successfully used for language representation learning in BERT [5] and many subsequent works, such as Albert [10], RoBERTa [9] and Electra [12]. It is observed good generalization ability of these models for various downstream language tasks, such as question answering and reading comprehension. Masked Image Modeling (MIM) adopts similar mechanism where a partial observation of the image is given to ML models to reconstruct the missing content. Stacked denoising autoencoders [221] was originally proposed for denoising corrupted image inputs using autoencoders. Context Encoders [222] masks arbitrary image regions and trains a convolutional neural network to predict missing parts based on context.

Recently, many Transformer-based models have been proposed for this purpose. For example, iGPT [219] learns a Transformer decoder to generate the next image pixel based on previously seen image pixels, while BEiT [220] utilizes an autoencoder to learn the latent space of discrete visual tokens and a BERT-like model for predicting visual labels for masked image patches, conditioned on unmasked patches.

7.2.2 Joint Visual-Linguistic Modeling

To facilitate joint modeling of visual and linguistic information, previous works, such as ViLBERT [211], LXMERT [212], and Pixel-BERT [213], typically use separate encoders of different architectures (*e.g.*, Convolutional Neural Network, and Transformers) to extract image and text features, and then global attention-based layers to fuse the features of the two modalities. Due to the difficulty of training different models in the same framework, and inspired by the success of Transformers in NLP and language modeling, a growing line of work adopts a unified Transformer model to jointly model image and text. Representative models include VL-BERT [215], Visual-BERT [216], Unicoder-VL [217], UNITER [218], and Oscar [200]. However, even using a unified Transformer in these models, there are still difficulties in extracting image features, because the image modalities lack visual analogies of vocabulary for efficient semantic decomposition. To alleviate this problem, most methods employ external detection models, such as Faster R-CNN, to extract image regions of interest (RoIs) and corresponding features. But Faster R-CNNs are mostly kept fixed (not trained) in these frameworks, preventing the models from reaching peak performance and robustness. To alleviate this problem and facilitate better image encoding, ViLT [223] represents images as patches [16], which can be directly fed into the model without additional pre-processing such as detection and segmentation.

Apart from differences among model architectures, various pre-training objectives have been proposed and proven effective for learning the cross-modality semantics, including but not limit to masked language modeling [213, 216, 217, 218, 200, 223], image-text matching [213, 214, 223], masked region modeling [214, 218], masked segment modeling [214], sentence-image prediction [216], object masked prediction [217], word region alignment [218], and contrastive loss [200, 224, 210]. From these, it has been observed that masked language modeling loss is extensively applied to facilitate language learning while matching-like losses such as image-text matching and contrastive loss are widely used for image semantic learning.

7.2.3 Autoencoders

Autoencoders [225] leverage an encoder to project input to a latent space and a decoder to reconstruct the input based on the latent representation. This task encourages the ML models to learn an effective latent space where features contain sufficient image information for reconstruction. Denoising autoencoders (DAE) [226] further corrupt the input to enhance model robustness. Recently, the rationale of auto-encoding is widely applied in visual and language representation learning, such as masked language modeling [213, 216, 217, 200] and masked image modeling [221, 222, 219, 220]. Masked Autoencoders [14] move a step further to adopt an autoencoder implemented by the Transformer to learn visual representation by reconstructing missing image pixels, and are proven effective when transferred to downstream image classification task. In the language domain, OPTIMUS [19] considers a Transformer-based Variational Autoencoder to learn language representation, which demonstrates effectiveness in guided language generation from an abstract level using the latent vectors and anjoy better generalize on low-resource language understanding tasks due to their smooth latent space structure.

7.3 Approach

7.3.1 Preliminaries

Masked Autoencoders (MAEs) [14] were proposed to learn visual representations in a self-supervised manner with an effective masked image modeling pre-text task. Specifically, images are fed to an autoencoder employing a Transformer-based encoder and decoder. Image patches are randomly masked and discarded during encoding and are only considered during decoding, whose output is used to reconstruct the missing image patches. MAE holds several technical merits:

1. It has been found that having a fairly high masking rate, *e.g.* 75%, helps MAE learn transferable visual representations [14].

2. An asymmetric encoder-decoder architecture that keeps the decoder lightweight is sufficient to achieve good performance on downstream tasks. Experimental results show that a 12-layer encoder and 8-layer decoder perform best in achieving linear probing and fine-tuning results on downstream ImageNet classification tasks [14].

3. The masked image patches do not need to be encoded by the encoder, which greatly speeds up the computation in the encoding process and improves the scalability of the model [14].

Inspired by the strong similarity between masked image modeling (MIM) tasks aimed at learning transferable visual representations and masked language modeling (MLM) tasks aimed at learning transferable language representations, we explore whether combining these two tasks can improve joint visual-linguistic modeling. In addition, MAEs do not exhibit any zero-shot image classification ability. However,

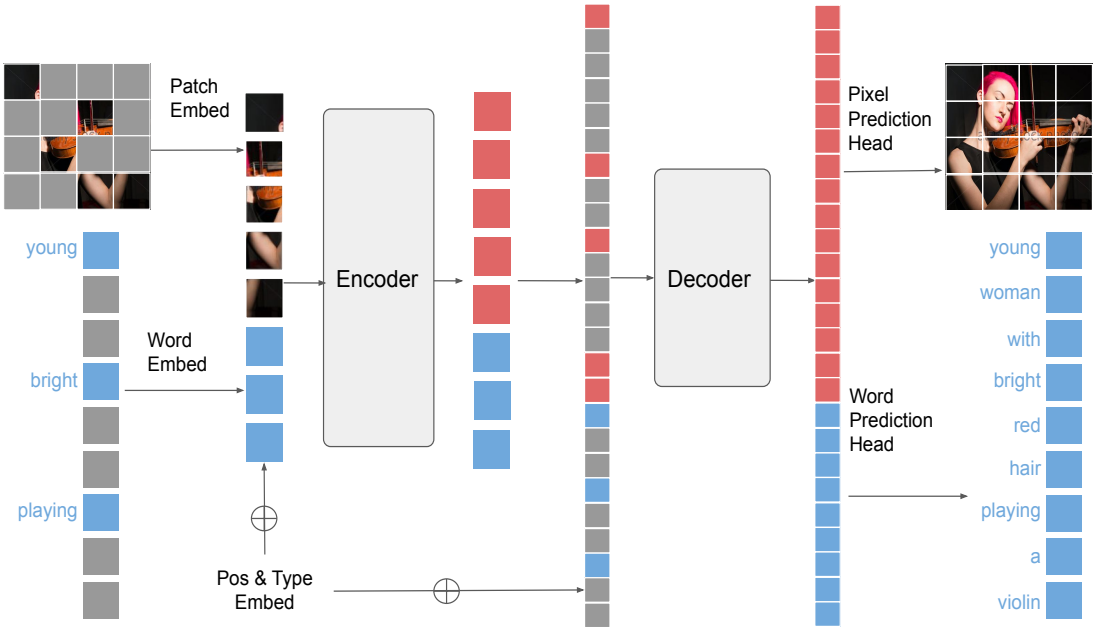


Figure 7.1: Proposed model architecture. Given an image-text pair, we set a fixed masking ratio to randomly mask image patches and word tokens. Then, we convert the visible image patches and word tokens into patch embeddings and word embeddings, respectively. Then, the visual and linguistic embeddings are concatenated, combined with positional embeddings and type embeddings, and fed to a unified Transformer-based encoder for joint visual-linguistic modeling. The output features of the encoder are added back with the masked components before passing them into a unified Transformer-based decoder. At the output of the decoder, pixel and word prediction heads are applied to predict masked image patches and word tokens, respectively

the coordination of MIM and MLM seeks enabling the proposed approach (VL-MAE) to have this capability, as additional language supervision is injected into pre-training. Further, choosing different content to mask may play an important role in learning well-transferred visual-linguistic features. Below we will examine these strategies in detail.

7.3.2 Masked Visual-Linguistic Modeling (MVLM)

We start by describing the proposed masked visual-linguistic modeling (MVLM) task which bridges the masked language modeling (MLM) task in the NLP domain

and the masked image modeling (MIM) task in the vision domain. MLM is widely adopted in Transformer-based language encoding [5, 9, 10, 12] where certain portion of the input tokens are randomly masked before feeding into a Transformer encoder and are reconstructed by a classifier appended to the encoder. Following a similar mechanism, MIM [14, 221, 222, 220] randomly masks some image patches obtained by segmenting the input image according to a predefined patch size in a non-overlap manner, while leveraging a classifier or a decoder to reconstruct the pixels of the missing patches. Inspired by the similarity between these two mechanisms, we propose masked visual-linguistic modeling (MVLM), which reconciles cross-domain visual and linguistic modeling in a mask-and-reconstruct manner.

Figure 7.1 illustrates the proposed model. Specifically, given an image-text pair, we first utilize a patch embedding layer to convert the image into a sequence of patch embeddings, and a word embedding layer to convert the input tokens into a sequence of word embeddings. Then, different masking mechanisms are applied to the image and text embeddings, respectively, which will be explained below. The visible image patches and text tokens are then collected and concatenated before feeding into a unified encoder for joint modeling. At the output of the encoder, the embeddings of mask tokens representing the missing image patches and text tokens are combined with the encoded latent features of the visible patches and tokens. Lastly, a light-weight decoder is utilized to take features of the visible and masked patches and tokens to reconstruct the original ones.

7.3.3 Visual-Linguistic Masked Autoencoders (VL-MAEs)

Complementary to the proposed MVLM task, several model design choices are carefully adopted and leveraged to help VL-MAE bridge the visual and linguistic domain gap

via a unified encoder-decoder architecture.

Masking

We apply different masking mechanisms to images and text sequences as distinct observations and insights are reported in literature for the two domains. For masked image modeling, it is understood that having a substantially high masking ratio, *e.g.*, 75%, in MAEs is beneficial for efficient and scalable visual representation learning [14], as the encoder processes only 25% of the image input. Therefore, we adopt 75% masking ratio for image patches as we resort to the same model architecture as in MAEs. Notably, the image patches are uniformly chosen at random for masking. For masked language modeling, while extensive BERT-like methods [5, 9, 10] promote low masking ratio, *e.g.*, 15%, we employ a fixed number of tokens not to be masked to improve computational efficiency, that is, we always select a fixed number of tokens to be passed into the encoder while masking the remaining tokens. In this way, we avoid the need to process padded tokens during encoding therefore boosting scalability.

Mask Tokens

The original MAE [14] incorporates a mask token to represent masked image patches during decoding. Features of masked tokens are then used to reconstruct original image patches at the last projection layer. Considering that the image and language are two different modalities, we employ separate mask tokens to represent masked image patches and masked text tokens, in order to help the decoder predict different kinds of output for different modalities.

Positional Embeddings

As images constitute two-dimensional information, we adopt a two-dimensional positional embeddings initialized by 2D sine-cosine position embedding weights to record image patches location. For one-dimensional text data, we use a one-dimensional embeddings initialized by 1D sine-cosine position embedding weights. Both positional embeddings start from zero even though the text features are placed after the image features during encoding. We found through experimentation that the starting positions do not have a substantial impact on performance. Both positional embedding weights are trainable during pre-training.

Type Embeddings

We further add type embeddings to the visual and linguistic features to indicate different modalities. All image patch features are assigned with the first type embeddings while all text token features are assigned with the second type embeddings. Type embeddings are summed up to the patch and token features before passing into the encoder and the decoder, respectively.

7.4 Experiments and Results

We pre-train VL-MAEs on four image-text datasets, including Google Conceptual Captions [227], Microsoft COCO (MSCOCO) [228], Visual Genome (VG) [229], and SBU Captions (SBU) [230]. After removing url links that do not result in downloaded valid images, we obtain around 8.62M samples in total for pre-training. Table 7.1 summarizes the dataset basic statistics. We pre-train both VL-MAE using 70% image masking ratio and 3 visible text tokens.

Table 7.1: Pre-training dataset summary statistics.

Dataset	# Images	# Captions	# Caption Tokens
Google Conceptual Captions (GCC)	3.01M	3.01M	10.66 ± 4.93
Microsoft COCO (MSCOCO)	0.113M	0.567M	11.81 ± 2.81
Visual Genome (VG)	0.108M	5.41M	5.53 ± 1.76
SBU Captions (SBU)	867K	867K	15.0 ± 7.74

A photo of fire | A photo of fog | A photo of room | A photo of woman | A photo of house | A photo of flag | A photo of traffic

Figure 7.2: Predicting image categories in a zero-shot manner using prompts.

7.4.1 Prompt Engineering

Due to the nature of VL-MAE's architecture, it can be conveniently transferred to zero-shot image classification task using prompts [8, 210]. Specifically, we feed an image and a prompt such as "`a photo of [MASK]`" as input to VL-MAEs to predict the "`[MASK]`" token whose output can be treated as the predicted category. Note that the output can be any word in the vocabulary. We visualize such straight-forward prediction in Figure 7.2. It is observed that VL-MAE is able to predict the major content in the images.

7.4.2 Zero-Shot Classification

Since the `[MASK]` token can only be used to predict one word, VL-MAEs may not infer the most comprehensive category that the image belongs to. To alleviate this issue, we further examine VL-MAE's zero-shot image classification capability on the ImageNet dataset using its 1000 labels as the output vocabulary. Specifically, we map the predictions of the `[MASK]` token to ImageNet labels by ranking them according to

Table 7.2: Cosine similarity between predicted top-5 class embeddings and ground-truth class embeddings, averaged for each image class in ImageNet.

Model	Cosine Sim
Supervised	0.7635
VL-MAE (Zero-shot)	0.7184

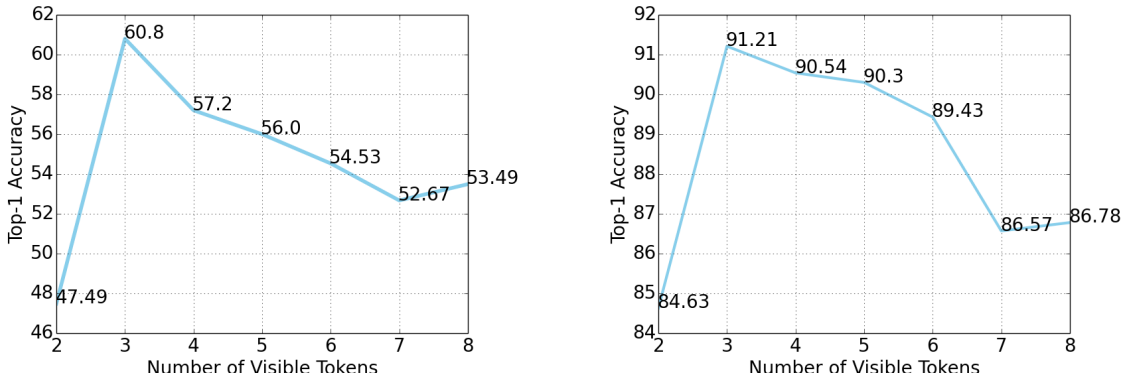
the average of their predicted probability for each token. For example, the predicted probability of the label "vine snake" is calculated as the average of the predicted probabilities of "vine" and "snake", respectively. Since VL-MAE sees millions of image captions during pre-training, we assumed that the model had some ability to predict class labels without any fine-tuning. Specifically, we tested our assumptions on the ImageNet dataset. Note that there are many rare classes in the ImageNet dataset, *e.g.*, sarong, hammerhead, Vizsla, kuvasz, and Yellow Lady's Slipper, that may not contain enough or any samples in VL-MAE's pretraining data. Particularly, Vizsla and Yellow Lady's Slipper have 2 and 0 samples in the Concept Caption dataset, respectively. If VL-MAE does not see enough images tagged with these categories, it may not be able to predict images belonging to these categories as the exact same words. However, it may be able to find other common categories that are similar, or parent categories to the ground truth category. Therefore, for each ImageNet dataset category, we collect the top 5 predicted categories for each image of that category and draw a word cloud to represent word frequencies.

Figure 7.3 shows these word visualizations. It is observed that VL-MAE predicts "dog", "hunting" and other similar words for the Vizsla category in Figure 7.3 (f), indicating that it learns very similar embeddings for the image Vizsla and these words in the embedding space. Likewise, gyroscopes, trolleybuses, sleds, little blue herons, hard drives, and mud turtles have no captions in the Conceptual Captions dataset. However, VL-MAE was able to predict very close keywords, such as "snowmobile"

Figure 7.3: Zero-shot word clouds. For every ImageNet category, we collect the top 5 categories predicted by VL-MAE for all images in the category ($\sim$ 1K images), and visualize word clouds reflecting the frequency of the predicted category tokens.

for "bobsled", "dog" for "borzoi", "mushrooms" for "gyromitra", "mud" and "turtle" for "mud turtle", *etc.* Moreover, categories in Figure 7.3 (g-p) have no samples in the entire pre-training dataset consisting of $\sim$ 2.65M samples. Importantly, VL-MAE successfully understands these classes of images without any fine-tuning, illustrating its strong generalization ability for zero-shot image classification tasks.

Further, we also quantify VL-MAE's zero-shot image classification ability by calculating the cosine similarity scores between the top-5 predicted class labels and



(a) Linear probing accuracy **(b)** Fine-tuning accuracy

Figure 7.4: Influence of the number of visible tokens on linear probing and fine-tuning performance on CIFAR10 dataset. Top-1 accuracy on test set is reported.

the ground truth labels, averaged over the dataset. Table 7.2 summarizes the results. Compared to the supervised baseline which specifically trains the model to maximize the cosine similarity between predicted labels and ground truth labels, VL-MAE achieves reasonably good results without seeing any ground truth labels of images in the dataset, demonstrating good zero-shot image classification transfer ability.

7.4.3 Ablation Study on Text Masking Ratio

We study the effect of the number of visible tokens on the downstream linear probing and fine-tuning image classification tasks. We pre-train VL-MAE on the COCO2014 [228] dataset, which is much smaller than the pre-training dataset. We further transfer the pre-trained model to CIFAR10 classification dataset and report top-1 accuracy. Since our VL-MAE sees massive amount of captions during pre-training, it should be applicable to different datasets with distributional shift. Therefore, we consider the small-size datasets to speed up the experiments. We study different numbers of visible text tokens from 2 to 8, which correspond to $0.83, 0.75, 0.66, 0.58, 0.49, 0.41, 0.32$ average masking ratios, computed by $1 - \frac{m}{l}$ where

m is the number of visible tokens and l is average length of captions in the dataset (see Table 7.1). We run 40 pre-training epochs, 5 warm-up epochs, 90 linear probing epochs, and 100 fine-tuning epochs.

Figure 7.4(a) illustrates the linear probing results. Interestingly, having just three visible tokens during pre-training leads to the best result on the downstream linear probing task. Having more visible tokens can in turn impair the encoder's ability to learn transferable visual features. We suspect this is because the encoder needs to delineate more of the missing content based on a few visible tokens, and thus learns better visual prediction ability. Moreover, having fewer visible tokens is closer to downstream image classification settings where no text is provided, resulting in stronger transfer performance. Figure 7.4(b) illustrates the fine-tuning results. Similar to the linear probing results, the best fine-tuning result is achieved using three visible tokens. Adding more visible tokens gives slightly worse results. However, using too many, *e.g.*, 7 and 8, or too few, *e.g.*, 2, visible tokens can greatly hurt fine-tuning performance.

7.5 Conclusion

In this chapter, we investigate the problem of jointly learning visual-linguistic representations with a unified Transformer architecture. Substantial amount of existing methods rely on a separate detection model such as Faster R-CNN to extract image regions and corresponding features, which prevents training from being end-to-end or ignore the consideration of global relations among different parts of the image. To alleviate this problem, we propose an efficient VL-MAE architecture that employs a unified Transformer model to jointly encode visual and linguistic features. We further propose Masked Visual Language Modeling (MVLM) task which bridges image and

text encoding in a mask-and-reconstruct manner. Extensive experiments on the ImageNet dataset showed that the proposed model achieves good results when transferred to zero-shot image classification, suggesting that our models have effectively learned the alignment between visual concepts and textual tokens without any supervision.

Chapter 8

Conclusion and Discussion

In this dissertation research proposal, we have made substantial and significant progress toward solving two critical issues associated with the Transformer, namely learning the expert knowledge and universal knowledge. First, we propose a novel Graph Transformer to learn medical knowledge graphs for medical image report generation and an automatic synchronization strategy searching framework for scalable and composable data-parallel distributed training. Second, we participated in developing a large-scale VAE to learn structured language representation via a smooth latent space, facilitating controlled generation of plausible text. Moreover, we propose learning commonsense reasoning by two effective SSL methods. Finally, we study applying Transformers to high-dimensional image input and learning meaningful image representations with extra language supervision. In pre-training, we may be faced with the need to scale up model size and training size. We propose to use the composable and automatic distributed training library—AutoDist [26] to accelerate model training and convergence. Finally, we will release our code to benefit the research community, share lessons learned to advance research in these fields, facilitate the application of Transformer models in expert domains, and make large-scale pre-trained models accessible to software developers who do not have expertise in ML and society as a whole.

Bibliography

[1] B. Dhingra, H. Liu, Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Gated-attention readers for text comprehension," 2016.

[2] S. Arora, H. Khandeparkar, M. Khodak, O. Plevrakis, and N. Saunshi, "A theoretical analysis of contrastive unsupervised representation learning," *arXiv preprint arXiv:1902.09229*, 2019.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.

[4] Y. Li, C. Li, Y. Zhang, X. Li, G. Zheng, L. Carin, and J. Gao, "Complementary auxiliary classifiers for label-conditional text generation.," in *AAAI*, pp. 8303–8310, 2020.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *NAACL*, 2018.

[6] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[7] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, 2019.

[8] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[9] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *ariv preprint arXiv:1907.11692*, 2019.

[10] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *ICLR*, 2019.

[11] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *NeurIPS*, pp. 5754–5764, 2019.

[12] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[13] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[14] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," *arXiv preprint arXiv:2111.06377*, 2021.

[15] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (bit): General visual representation learning," *arXiv preprint arXiv:1912.11370*, 2019.

[16] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[17] M. Caron, P. Bojanowski, J. Mairal, and A. Joulin, "Unsupervised pre-training of image features on non-curated data," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2959–2968, 2019.

[18] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," *arXiv preprint arXiv:2002.05709*, 2020.

[19] C. Li, X. Gao, Y. Li, X. Li, B. Peng, Y. Zhang, and J. Gao, "Optimus: Organizing sentences via pre-trained modeling of a latent space," *arXiv preprint arXiv:2004.04092*, 2020.

[20] T. H. Trinh and Q. V. Le, "A simple method for commonsense reasoning," *arXiv preprint arXiv:1806.02847*, 2018.

[21] H. Liu and P. Singh, "Conceptnet—a practical commonsense reasoning tool-kit," *BT technology journal*, vol. 22, no. 4, pp. 211–226, 2004.

[22] T. Chen, L. Zheng, E. Yan, Z. Jiang, T. Moreau, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Learning to optimize tensor programs," in *Advances in Neural Information Processing Systems*, pp. 3389–3400, 2018.

[23] C. Y. Li, X. Liang, Z. Hu, and E. P. Xing, "Hybrid retrieval-generation reinforced agent for medical image report generation," *NIPS*, 2018.

[24] C. Y. Li, X. Liang, Z. Hu, and E. P. Xing, "Knowledge-driven encode, retrieve, paraphrase for medical image report generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 6666–6673, 2019.

[25] Y. Yang, P. Xie, X. Gao, C. Cheng, C. Li, H. Zhang, and E. Xing, "Predicting discharge medications at admission time based on deep learning," *arXiv preprint arXiv:1711.01386*, 2017.

[26] H. Zhang, Y. Li, Z. Deng, X. Liang, L. Carin, and E. Xing, "Autosync: Learning to synchronize for data-parallel distributed deep learning," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[27] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed dnn training acceleration," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 16–29, 2019.

[28] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 583–598, 2014.

[29] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[30] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on {GPU} clusters," in *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pp. 181–193, 2017.

[31] S. Kim, G.-I. Yu, H. Park, S. Cho, E. Jeong, H. Ha, S. Lee, J. S. Jeong, and B.-G. Chun, "Parallax: Automatic data-parallel training of deep neural networks," *arXiv preprint arXiv:1808.02621*, 2018.

[32] Y. Li, B. Yu, M. Xue, and T. Liu, "Enhancing pre-trained chinese character representation with word-aligned attention," *ACL*, 2019.

[33] B. McCann, J. Bradbury, C. Xiong, and R. Socher, "Learned in translation: Contextualized word vectors," 2017.

[34] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H.-W. Hon, "Unified language model pre-training for natural language understanding and generation," *NeurIPS*, 2019.

[35] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv preprint arXiv:1910.10683*, 2019.

[36] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision*, pp. 213–229, Springer, 2020.

[37] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," *arXiv preprint arXiv:2103.14030*, 2021.

[38] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," *arXiv preprint arXiv:2104.14294*, 2021.

[39] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, "Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server," in *Proceedings of the Eleventh European Conference on Computer Systems*, p. 4, ACM, 2016.

[40] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[41] P. Xie, J. K. Kim, Y. Zhou, Q. Ho, A. Kumar, Y. Yu, and E. Xing, "Distributed Machine Learning via Sufficient Factor Broadcasting," in *arXiv*, 2015.

[42] Z. Jia, S. Lin, C. R. Qi, and A. Aiken, "Exploring hidden dimensions in parallelizing convolutional neural networks," *arXiv preprint arXiv:1802.04924*, 2018.

[43] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *ICML*, 2015.

[44] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *CVPR*, 2015.

[45] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel, "Self-critical sequence training for image captioning," in *CVPR*, 2017.

[46] X. Liang, Z. Hu, H. Zhang, C. Gan, and E. P. Xing, "Recurrent topic-transition gan for visual paragraph generation," in *ICCV*, 2017.

[47] T.-H. K. Huang, F. Ferraro, N. Mostafazadeh, I. Misra, A. Agrawal, J. Devlin, R. Girshick, X. He, P. Kohli, D. Batra, *et al.*, "Visual storytelling," in *ACL*, 2016.

[48] J. Krause, J. Johnson, R. Krishna, and L. Fei-Fei, "A hierarchical approach for generating descriptive image paragraphs," in *CVPR*, 2017.

[49] J. Lu, J. Yang, D. Batra, and D. Parikh, "Neural baby talk," in *CVPR*, 2018.

[50] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, "Bottom-up and top-down attention for image captioning and vqa," in *CVPR*, 2018.

[51] B. Taskar, C. Guestrin, and D. Koller, "Max-margin markov networks," in *NIPS*, 2004.

[52] Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing, "Harnessing deep neural networks with logic rules," in *ACL*, 2016.

[53] E. Strubell, P. Verga, D. Andor, D. Weiss, and A. McCallum, "Linguistically-informed self-attention for semantic role labeling," in *EMNLP*, 2018.

[54] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, *et al.*, "Never-ending learning," *Communications of the ACM*, 2018.

[55] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data: The story so far," in *Semantic services, interoperability and web applications: emerging concepts*, 2011.

[56] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, "Relational inductive biases, deep learning, and graph networks," *NIPS*, 2017.

[57] X. Liang, Z. Hu, and E. Xing, "Symbolic graph reasoning meets convolutions," in *NIPS*, 2018.

[58] Z. Hu, Z. Yang, R. Salakhutdinov, X. Liang, L. Qin, H. Dong, and E. Xing, "Deep generative models with learnable knowledge constraints," *NIPS*, 2018.

[59] A. G. X. Wang, Y. Ye, "Zero-shot recognition via semantic embeddings and knowledge graphs," *CVPR*, 2018.

[60] B. Jing, P. Xie, and E. Xing, "On the automatic generation of medical imaging reports," *ACL*, 2018.

[61] Y. Hong and C. E. Kahn, "Content analysis of reporting templates and free-text radiology reports," *JDI*, 2013.

[62] D. Demner-Fushman, M. D. Kohli, M. B. Rosenman, S. E. Shooshan, L. Rodriguez, S. Antani, G. R. Thoma, and C. J. McDonald, "Preparing a collection of radiology examinations for distribution and retrieval," *JAMIA*, 2015.

[63] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *CVPR*, 2017.

[64] X. Wang, Y. Peng, L. Lu, Z. Lu, and R. M. Summers, "Tienet: Text-image embedding network for common thorax disease classification and reporting in chest x-rays," in *CVPR*, 2018.

[65] Z. Li, C. Wang, M. Han, Y. Xue, W. Wei, L.-J. Li, and F. Li, "Thoracic disease identification and localization with limited supervision," *CVPR*, 2017.

[66] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016.

[67] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR*, 2017.

[68] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *NIPS*, 2017.

[69] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," *CVPR*, 2018.

[70] N. Parmar, A. Vaswani, J. Uszkoreit, Ł. Kaiser, N. Shazeer, and A. Ku, "Image transformer," *ICLR*, 2018.

[71] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *ICLR*, 2018.

[72] S. L. Ziqiang Cao, Wenjie Li and F. Wei, "Retrieve, rerank and rewrite: Soft template based neural summarization," in *ACL*, 2018.

[73] A. Gupta, A. Agarwal, P. Singh, and P. Rai, "A deep generative framework for paraphrase generation," *AAAI*, 2018.

[74] K. Guu, T. B. Hashimoto, Y. Oren, and P. Liang, "Generating sentences by editing prototypes," *AAAI*, 2018.

[75] J. Mallinson, R. Sennrich, and M. Lapata, "Paraphrasing revisited with neural machine translation," in *ACL*, 2017.

[76] ""jieba" (chinese for "to stutter") chinese text segmentation: built to be the best python chinese word segmentation module.." Accessed: 2018-05-01.

[77] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *CVPR*, 2017.

[78] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection.," in *CVPR*, 2017.

[79] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, "Cider: Consensus-based image description evaluation," in *CVPR*, 2015.

[80] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *ACL*, 2013.

[81] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *ACL*, 2002.

[82] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *CVPR*, 2015.

[83] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *CVPR*, 2015.

[84] J. Lu, C. Xiong, D. Parikh, and R. Socher, "Knowing when to look: Adaptive attention via a visual sentinel for image captioning," in *CVPR*, 2017.

[85] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[86] T. Chilimbi, Y. S. J. Apacible, and K. Kalyanaraman, "Project Adam: Building an Efficient and Scalable Deep Learning Training System," in *OSDI*, 2014.

[87] J. Wei, W. Dai, A. Qiao, Q. Ho, H. Cui, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Managed Communication and Consistency for Fast Data-parallel Iterative Analytics," in *SoCC*, 2015.

[88] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, pp. 173–182, 2017.

[89] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[90] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2430–2439, JMLR. org, 2017.

[91] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean, "A hierarchical model for device placement," 2018.

[92] A. Paliwal, F. Gimeno, V. G. Nair, Y. Li, M. Lubin, P. Kohli, and O. Vinyals, "Reinforced genetic algorithm learning for optimizing computation graphs," 2020.

[93] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, *et al.*, "{TVM}: An automated end-to-end optimizing compiler for deep learning," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 578–594, 2018.

[94] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[95] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," *arXiv preprint arXiv:1807.05358*, 2018.

[96] M. Wang, C.-c. Huang, and J. Li, "Supporting very large models using automatic dataflow graph partitioning," in *Proceedings of the Fourteenth EuroSys Conference 2019*, p. 26, ACM, 2019.

[97] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[98] H. Xu, H. Zhang, Z. Hu, X. Liang, R. Salakhutdinov, and E. Xing, "Autoloss: Learning discrete schedules for alternate optimization," *arXiv preprint arXiv:1810.02442*, 2018.

[99] P. Xie, J. K. Kim, Q. Ho, Y. Yu, and E. Xing, "Orpheus: Efficient distributed machine learning via system and algorithm co-design," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 1–13, 2018.

[100] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel dnn training," *arXiv preprint arXiv:1806.03377*, 2018.

[101] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, "Tensorflow: A system for large-scale machine learning," *arXiv preprint arXiv:1605.08695*, 2016.

[102] Facebook. `http://pytorch.org/`.

[103] N. nccl tests, "Performance reported by nccl tests," 2018.

[104] G. Wang, S. Venkataraman, A. Phanishayee, J. Thelin, N. Devanur, and I. Stoica, "Blink: Fast and generic collectives for distributed ml," *arXiv preprint arXiv:1910.04940*, 2019.

[105] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," *arXiv preprint arXiv:1803.07294*, 2018.

[106] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[107] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of the 24th international conference on Machine learning*, pp. 129–136, 2007.

[108] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[109] A. Krause and D. Golovin, "Submodular function maximization.."

[110] Google, "Distributed tensorflow," *https://github.com/tensorflow/examples/blob/master/community/en/docs/deploy/distributed.md*.

[111] Q. Ho, J. Cipar, H. Cui, J. K. Kim, S. Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing, "More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server," in *NIPS*, 2013.

[112] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[113] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *arXiv preprint arXiv:1512.00567*, 2015.

[114] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, *et al.*, "Mlperf training benchmark," *arXiv preprint arXiv:1910.01500*, 2019.

[115] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.

[116] M. Li, D. G. Andersen, A. Smola, and K. Yu, "Communication Efficient Distributed Machine Learning with the Parameter Server," in *NIPS*, 2014.

[117] Medium *https://medium.com/future-vision/bert-meets-gpus-403d3fbed848*.

[118] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," 2018.

[119] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher, "Ctrl: A conditional transformer language model for controllable generation," *arXiv preprint arXiv:1909.05858*, 2019.

[120] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using gpu model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[121] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *ICLR*, 2013.

[122] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *ICML*, 2014.

[123] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," *CONLL*, 2016.

[124] Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing, "Toward controlled generation of text," *arXiv preprint arXiv:1703.00955*, 2017.

[125] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[126] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-thought vectors," in *NIPS*, 2015.

[127] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *NIPS*, 2015.

[128] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *NAACL*, 2019.

[129] Z. Yang, Z. Hu, C. Dyer, E. P. Xing, and T. Berg-Kirkpatrick, "Unsupervised text style transfer using language models as discriminators," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[130] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, L. Carin, *et al.*, "Cyclical annealing schedule: A simple approach to mitigating KL vanishing," *NAACL*, 2019.

[131] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," *ICLR*, 2017.

[132] B. Li, J. He, G. Neubig, T. Berg-Kirkpatrick, and Y. Yang, "A surprisingly effective fix for deep latent variable modeling of text," *EMNLP*, 2019.

[133] Z. Yang, Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick, "Improved variational autoencoders for text modeling using dilated convolutions," *ICML*, 2017.

[134] A. B. Dieng, Y. Kim, A. M. Rush, and D. M. Blei, "Avoiding latent variable collapse with generative skip models," *AISTATS*, 2018.

[135] T. Zhao, R. Zhao, and M. Eskenazi, "Learning discourse-level diversity for neural dialog models using conditional variational autoencoders," *ACL*, 2017.

[136] Y. Kim, S. Wiseman, A. C. Miller, D. Sontag, and A. M. Rush, "Semi-amortized variational autoencoders," *ICML*, 2018.

[137] J. He, D. Spokoyny, G. Neubig, and T. Berg-Kirkpatrick, "Lagging inference networks and posterior collapse in variational autoencoders," *ICLR*, 2019.

[138] L. Fang, C. Li, J. Gao, W. Dong, and C. Chen, "Implicit deep latent variable models for text generation," *EMNLP*, 2019.

[139] J. Zhao, Y. Kim, K. Zhang, A. M. Rush, and Y. LeCun, "Adversarially regularized autoencoders," *ICML*, 2018.

[140] S. Subramanian, S. R. Mudumba, A. Sordoni, A. Trischler, A. C. Courville, and C. Pal, "Towards text generation with adversarially learned neural outlines," in *NeurIPS*, 2018.

[141] Y. Zhu, S. Lu, L. Zheng, J. Guo, W. Zhang, J. Wang, and Y. Yu, "Texygen: A benchmarking platform for text generation models," in *ACM SIGIR*, 2018.

[142] T. Shen, T. Lei, R. Barzilay, and T. Jaakkola, "Style transfer from non-parallel text by cross-alignment," in *NIPS*, 2017.

[143] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[144] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh annual conference of the international speech communication association*, 2010.

[145] J. Zhao, Y. Kim, K. Zhang, A. M. Rush, and Y. LeCun, "Adversarially regularized autoencoders," *arXiv preprint arXiv:1706.04223*, 2017.

[146] J. Li, R. Jia, H. He, and P. Liang, "Delete, retrieve, generate: A simple approach to sentiment and style transfer," *arXiv preprint arXiv:1804.06437*, 2018.

[147] J. Xu, X. Sun, Q. Zeng, X. Ren, X. Zhang, H. Wang, and W. Li, "Unpaired sentiment-to-sentiment translation: A cycled reinforcement learning approach," *arXiv preprint arXiv:1805.05181*, 2018.

[148] Y. Zhang, N. Ding, and R. Soricut, "Shaped: Shared-private encoder-decoder for text style adaptation," *arXiv preprint arXiv:1804.04093*, 2018.

[149] A. A. Alemi, B. Poole, I. Fischer, J. V. Dillon, R. A. Saurous, and K. Murphy, "Fixing a broken elbo," *arXiv preprint arXiv:1711.00464*, 2017.

[150] X. Liu, J. Gao, A. Celikyilmaz, L. Carin, *et al.*, "Cyclical annealing schedule: A simple approach to mitigating kl vanishing," *arXiv preprint arXiv:1903.10145*, 2019.

[151] X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel, "Variational lossy autoencoder," *arXiv preprint arXiv:1611.02731*, 2016.

[152] C. Li, H. Liu, C. Chen, Y. Pu, L. Chen, R. Henao, and L. Carin, "ALICE: Towards understanding adversarial learning for joint distribution matching," in *NIPS*, 2017.

[153] K. Shuster, S. Humeau, H. Hu, A. Bordes, and J. Weston, "Engaging image captioning via personality," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12516–12526, 2019.

[154] C. Gan, Z. Gan, X. He, J. Gao, and L. Deng, "Stylenet: Generating attractive visual captions with styles," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3137–3146, 2017.

[155] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, pp. 649–657, 2015.

[156] Z. Fu, X. Tan, N. Peng, D. Zhao, and R. Yan, "Style transfer in text: Exploration and evaluation," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[157] N. Dai, J. Liang, X. Qiu, and X. Huang, "Style transformer: Unpaired text style transfer without disentangled latent representation," *arXiv preprint arXiv:1905.05621*, 2019.

[158] S. Prabhumoye, Y. Tsvetkov, R. Salakhutdinov, and A. W. Black, "Style transfer through back-translation," *arXiv preprint arXiv:1804.09000*, 2018.

[159] J. G. W. D. C. C. Le Fang, Chunyuan Li, "Implicit deep latent variable models for text generation," *EMNLP*, 2019.

[160] R. Xu, T. Ge, and F. Wei, "Formality style transfer with hybrid textual annotations," *arXiv preprint arXiv:1903.06353*, 2019.

[161] P. Xu, Y. Cao, and J. C. K. Cheung, "Unsupervised controllable text generation with global variation discovery and disentanglement," *arXiv preprint arXiv:1905.11975*, 2019.

[162] Z. Zhang, S. Ren, S. Liu, J. Wang, P. Chen, M. Li, M. Zhou, and E. Chen, "Style transfer as unsupervised machine translation," *arXiv preprint arXiv:1808.07894*, 2018.

[163] F. Luo, P. Li, J. Zhou, P. Yang, B. Chang, Z. Sui, and X. Sun, "A dual reinforcement learning framework for unsupervised text style transfer," *arXiv preprint arXiv:1905.10060*, 2019.

[164] K. Wang, H. Hua, and X. Wan, "Controllable unsupervised text attribute transfer via editing entangled latent representation," *arXiv preprint arXiv:1905.12926*, 2019.

[165] N. Tandon, B. D. Mishra, K. Sakaguchi, A. Bosselut, and P. Clark, "Wiqa: A dataset for" what if..." reasoning over procedural text," *EMNLP*, 2019.

[166] L. Huang, R. L. Bras, C. Bhagavatula, and Y. Choi, "Cosmos qa: Machine reading comprehension with contextual commonsense reasoning," *EMNLP*, 2019.

[167] W. Yu, Z. Jiang, Y. Dong, and J. Feng, "Reclor: A reading comprehension dataset requiring logical reasoning," *ICLR*, 2020.

[168] J. Zhu, Y. Xia, L. Wu, D. He, T. Qin, W. Zhou, H. Li, and T.-Y. Liu, "Incorporating bert into neural machine translation," *ICLR*, 2020.

[169] C. Zhao, C. Xiong, C. Rosset, X. Song, P. Bennett, and S. Tiwary, "Transformer-xh: Multi-evidence reasoning with extra hop attention," in *ICLR*, April 2020.

[170] K. Ma, J. Francis, Q. Lu, E. Nyberg, and A. Oltramari, "Towards generalizable neuro-symbolic systems for commonsense question answering," *EMNLP*, 2019.

[171] B. Dhingra, M. Zaheer, V. Balachandran, G. Neubig, R. Salakhutdinov, and W. W. Cohen, "Differentiable reasoning over a virtual knowledge base," *arXiv preprint arXiv:2002.10640*, 2020.

[172] X. Zhan, Y. Li, X. Dong, X. Liang, Z. Hu, and L. Carin, "elberto: Self-supervised commonsense learning for question answering," *arxiv.org/pdf/2203.09424*, 2022.

[173] Y. Jernite, S. R. Bowman, and D. Sontag, "Discourse-based objectives for fast unsupervised sentence representation learning," *arXiv preprint arXiv:1705.00557*, 2017.

[174] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu, "ERNIE: Enhanced language representation with informative entities," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, (Florence, Italy), pp. 1441–1451, ACL, July 2019.

[175] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "Spanbert: Improving pre-training by representing and predicting spans," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77, 2020.

[176] Y. Sun, E. Tzeng, T. Darrell, and A. A. Efros, "Unsupervised domain adaptation through self-supervision," *arXiv preprint arXiv:1909.11825*, 2019.

[177] Y. Sun, X. Wang, Z. Liu, J. Miller, A. A. Efros, and M. Hardt, "Test-time training for out-of-distribution generalization," *arXiv preprint arXiv:1909.13231*, 2019.

[178] A. Asai, K. Hashimoto, H. Hajishirzi, R. Socher, and C. Xiong, "Learning to retrieve reasoning paths over wikipedia graph for question answering," *arXiv preprint arXiv:1911.10470*, 2019.

[179] Y. Huang, M. Fang, X. Zhan, Q. Cao, X. Liang, and L. Lin, "Rem-net: Recursive erasure memory network for commonsense evidence refinement," *AAAI*, 2020.

[180] S. Kundu, T. Khot, A. Sabharwal, and P. Clark, "Exploiting explicit paths for multi-hop reading comprehension," *ACL*, 2018.

[181] T. Mihaylov and A. Frank, "Knowledgeable reader: Enhancing cloze-style reading comprehension with external commonsense knowledge," *ACL*, 2018.

[182] M. Tu, G. Wang, J. Huang, Y. Tang, X. He, and B. Zhou, "Multi-hop reading comprehension across multiple documents by reasoning over heterogeneous graphs," *ACL*, 2019.

[183] Y. Huang, M. Fang, Y. Cao, L. Wang, and X. Liang, "Dagn: Discourse-aware graph network for logical reasoning," *NAACL*, 2021.

[184] M. S. Zahedi, M. Rahgozar, and R. A. Zoroofi, "HCA: hierarchical compare aggregate model for question retrieval in community question answering," *Inf. Process. Manag.*, vol. 57, no. 6, p. 102318, 2020.

[185] B. Y. Lin, X. Chen, J. Chen, and X. Ren, "Kagnet: Knowledge-aware graph networks for commonsense reasoning," *EMNLP*, 2019.

[186] A. Asai and H. Hajishirzi, "Logic-guided data augmentation and regularization for consistent question answering," *arXiv preprint arXiv:2004.10157*, 2020.

[187] W. Xiong, J. Du, W. Y. Wang, and V. Stoyanov, "Pretrained encyclopedia: Weakly supervised knowledge-pretrained language model," *ICLR*, 2019.

[188] Y. Sun, S. Wang, Y. Li, S. Feng, X. Chen, H. Zhang, X. Tian, D. Zhu, H. Tian, and H. Wu, "Ernie: Enhanced representation through knowledge integration," *ACL*, vol. abs/1904.09223, 2019.

[189] F. Zhu, Y. Zhu, X. Chang, and X. Liang, "Vision-language navigation with self-supervised auxiliary reasoning tasks," in *CVPR*, pp. 10009–10019, IEEE, 2020.

[190] V. Veeriah, M. Hessel, Z. Xu, R. Lewis, J. Rajendran, J. Oh, H. van Hasselt, D. Silver, and S. Singh, "Discovery of useful questions as auxiliary tasks," *arXiv preprint arXiv:1909.04607*, 2019.

[191] C.-Y. Ma, J. Lu, Z. Wu, G. AlRegib, Z. Kira, R. Socher, and C. Xiong, "Self-monitoring navigation agent via auxiliary progress estimation," *arXiv preprint arXiv:1901.03035*, 2019.

[192] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[193] L. Logeswaran, H. Lee, and D. Radev, "Sentence ordering and coherence modeling using recurrent neural networks," in *AAAI*, 2018.

[194] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NeurIPS*, pp. 2787–2795, 2013.

[195] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

[196] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *UAI*, 2018.

[197] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *ICLR*, 2018.

[198] T. Kipf, E. van der Pol, and M. Welling, "Contrastive learning of structured world models," *ICLR*, 2019.

[199] V. François-Lavet, Y. Bengio, D. Precup, and J. Pineau, "Combined reinforcement learning via abstract representations," in *AAAI*, vol. 33, pp. 3582–3589, 2019.

[200] X. Li, X. Yin, C. Li, P. Zhang, X. Hu, L. Zhang, L. Wang, H. Hu, L. Dong, F. Wei, *et al.*, "Oscar: Object-semantics aligned pre-training for vision-language tasks," in *European Conference on Computer Vision*, pp. 121–137, Springer, 2020.

[201] J. Li and E. Hovy, "A model of coherence based on distributed sentence representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2039–2048, 2014.

[202] J. Li and D. Jurafsky, "Neural net models for open-domain discourse coherence," *EMNLP*, 2016.

[203] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang, "Realm: Retrieval-augmented language model pre-training," *arXiv preprint arXiv:2002.08909*, 2020.

[204] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *European conference on computational learning theory*, pp. 23–37, Springer, 1995.

[205] L. Wang, M. Sun, W. Zhao, K. Shen, and J. Liu, "Yuanfudao at semeval-2018 task 11: Three-way attention and relational knowledge for commonsense machine comprehension," *ACL*, 2018.

[206] S. Zhang, H. Zhao, Y. Wu, Z. Zhang, X. Zhou, and X. Zhou, "Dual co-matching network for multi-choice reading comprehension," *AAAI*, 2019.

[207] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *EACL*, 2016.

[208] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, pp. 1532–1543, 2014.

[209] S. Min, D. Chen, L. Zettlemoyer, and H. Hajishirzi, "Knowledge guided text retrieval and reading for open domain question answering," 2020.

[210] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," *arXiv preprint arXiv:2103.00020*, 2021.

[211] J. Lu, D. Batra, D. Parikh, and S. Lee, "Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks," *arXiv preprint arXiv:1908.02265*, 2019.

[212] H. Tan and M. Bansal, "Lxmert: Learning cross-modality encoder representations from transformers," *arXiv preprint arXiv:1908.07490*, 2019.

[213] Z. Huang, Z. Zeng, B. Liu, D. Fu, and J. Fu, "Pixel-bert: Aligning image pixels with text by deep multi-modal transformers," *arXiv preprint arXiv:2004.00849*, 2020.

[214] J. Lin, A. Yang, Y. Zhang, J. Liu, J. Zhou, and H. Yang, "Interbert: Vision-and-language interaction for multi-modal pretraining," *arXiv preprint arXiv:2003.13198*, 2020.

[215] W. Su, X. Zhu, Y. Cao, B. Li, L. Lu, F. Wei, and J. Dai, "Vl-bert: Pre-training of generic visual-linguistic representations," *arXiv preprint arXiv:1908.08530*, 2019.

[216] L. H. Li, M. Yatskar, D. Yin, C.-J. Hsieh, and K.-W. Chang, "Visualbert: A simple and performant baseline for vision and language," *arXiv preprint arXiv:1908.03557*, 2019.

[217] G. Li, N. Duan, Y. Fang, M. Gong, and D. Jiang, "Unicoder-vl: A universal encoder for vision and language by cross-modal pre-training," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 11336–11344, 2020.

[218] Y.-C. Chen, L. Li, L. Yu, A. El Kholy, F. Ahmed, Z. Gan, Y. Cheng, and J. Liu, "Uniter: Learning universal image-text representations," *https://openreview.net/pdf?id=S1eL4kBYwr*, 2019.

[219] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," *arXiv preprint arXiv:2102.12092*, 2021.

[220] H. Bao, L. Dong, and F. Wei, "Beit: Bert pre-training of image transformers," *arXiv preprint arXiv:2106.08254*, 2021.

[221] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.," *Journal of machine learning research*, vol. 11, no. 12, 2010.

[222] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2536–2544, 2016.

[223] W. Kim, B. Son, and I. Kim, "Vilt: Vision-and-language transformer without convolution or region supervision," *arXiv preprint arXiv:2102.03334*, 2021.

[224] F. Luo, P. Yang, S. Li, X. Ren, and X. Sun, "Capt: Contrastive pre-training for learning denoised sequence representations," *arXiv preprint arXiv:2010.06351*, 2020.

[225] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and helmholtz free energy," *Advances in neural information processing systems*, vol. 6, pp. 3–10, 1994.

[226] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

[227] P. Sharma, N. Ding, S. Goodman, and R. Soricut, "Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2556–2565, 2018.

[228] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[229] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, *et al.*, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International journal of computer vision*, vol. 123, no. 1, pp. 32–73, 2017.

[230] V. Ordonez, G. Kulkarni, and T. Berg, "Im2text: Describing images using 1 million captioned photographs," *Advances in neural information processing systems*, vol. 24, 2011.

Biography

Yuan Li earned her Bachelor of Engineering with Honours, Second Class Upper Division from the Chinese University of Hong Kong in 2015 and her degree of Master of Computational Data Science from Carnegie Mellon University in 2017. She expects to receive her Doctor of Philosophy in Electrical/Computer Engg-PHD in 2022. Her research interests lay in Transformers, self-supervised learning, representation learning, distributed training, and medical diagnosis.