ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Data Mining, Text Mining and Big Data Analytics

NEXT PICTOGRAM PREDICTION VIA VISION-LANGUAGE MODELING: ENHANCING COMMUNICATION OF AUTISTIC CHILDREN

CANDIDATE
Martina Ianaro

SUPERVISOR
Prof. Gianluca Moro

CO-SUPERVISOR
Giacomo Frisoni, PhD.

Academic year 2023-2024

Session 2rd

Keywords

Large Language Model

Multimodal AI

Language modeling

Natural Language Processing

Text Mining

Computer Vision

Transformer

Deep Learning

Augmentative and alternative communication

Pictogram Prediction

Autism

Failure is success in progress.

Albert Einstein

Contents

List of Figures

List of Tables

Next Pictogram Prediction via Vision-Language Modeling: Enhancing Communication of Autistic Children

October 5, 2023

Abstract

Individuals with complex communication needs often face speech barriers that hinder social inclusion. As said by I. Estrada, *"If a child cannot learn the way we teach, maybe we should teach the way they learn"*. This philosophy guides our research, which focuses on empowering children with autism and other communication disorders to effectively utilize Augmentative and Alternative Communication Systems, better predicting their needs. Recent multimodal AI breakthroughs have opened up new avenues for enhancing the lives of these individuals. In this context, we introduce PictoViLT, a Vision-and-Language Transformer Encoder fine-tuned on text datasets and ARASAAC pictograms. Our approach involves employing various self-supervised masking techniques that transition from single to multiple text tokens, effectively addressing the challenge of predicting the next pictogram. PictoViLT breaks the strict dependence on WordNet concept sequences as input, being able to manage natural language and pictogram sequences directly. In-depth experiments conducted on various datasets, grounded on commonsense resources and verbalized knowledge graphs, reveal significant enhancements compared to prior state-of-the-art models. In contrast to PictoBERT and statistical n-gram models, PictoViLT achieves up to +0.60 Top-1 accuracy points. Ultimately, token–patch alignments and attention areas make predictions interpretable.

Introduction

Chapter 1

Enhancing Autism Communication via AAC Technology

The new millennium has seen an increase in awareness of autism due to more media attention and a rapidly growing body of knowledge published in professional journals.

Pediatricians must be able to detect signs of autism spectrum disorders, as well as a strategy to methodically evaluate them [20].

Autism spectrum disorder (ASD) is the broader spectrum of clinical characteristics that now define autism.

ASD is not rare; According to the article by Johnson et al. (2007) [20], many pediatricians care for several children, with 44% of them caring for at least 10 children. In 2007, the raging of 8-year-old children passed from 1 to 303 and became 1 in 94 at 14 sites in the USA.

We can speak about the ”autism epidemic”, one of the most challenging public health issues today.

Following years of research, autism first appeared in the Diagnostic and Statistical Manual of Mental Disorders DSM-III in 1980 and was modified in DSM-IV in 1994.

ASD is a neurodevelopmental condition with highly heritable biological roots, although its exact causes are not yet clear.

Social impairments, warning symptoms of ASD, are usually used as clinical signs to detect autism.

The idea is to identify the condition in the first years of life, but usually, symptoms remain unnoticed until the kid is in the classroom and the instructors observe difficulties with relationships with peers.

Poor social skills make it difficult to communicate emotions, while the absence of communication skills leads to ”speech delay”. No desire to communicate, and nonverbal corrective actions, such as gestures, are linked to the absence of speech. This time its exact causes remain mysterious.

About 30% of the children start speaking, then give up, followed by a decrease in skills.

The article by Almeida et al. [2], enhances the use of high-tech devices for Augmentative and Alternative Communication (AAC) in the home setting.

This study considers the research problem of how to promote assistive technology (AT) use by children with cognitive limitations.

This study aims to analyze children’s performance and their interaction with their caregivers to improve practices regarding the use of AT.

AAC systems must be customized to meet users’ needs, in terms of ease of use and learning, and each device must fit the age and personality of its users.

Due to the COVID-19 epidemic, 2020 and the years that followed were exceptionally challenging for children with disabilities, as adversity can come in multiple forms.

Distance learning and communication, whether due to scarce resources, novelty, isolation, or little or no digital literacy, made interacting and learning for most students challenging enough.

AT is likely to promote communication, autonomy, and inclusion of children with cognitive and/or motor disabilities. It opens a wide range of opportunities to learn and participate in all settings of life, as long as caregivers have an active role in the implementation of AAC devices and the promotion of their regular use at home.

The failure to provide appropriate AAC technology, strategies, and services means a loss of opportunities for the individual and for society. There are many strategies that educators use to help a child communicate.

Visual aids have been used successfully to help children with autism communicate.

A picture system called AAC Media allows teachers to take pictures to help students in the spectrum communicate.

In the work by Erlani et al.(2018) [11], the use of AAC media has a positive impact on the skills of autistic students in musical dynamics.

As described by Zhao et al.(2023)[58], proposed a Mixed Reality-based wearable AAC device that shows high usability.

The results of the study by Andzik et al. [3] demonstrate that paraeducators also need to be prepared to help children with CCN.

Before the intervention, the student was rarely observed initiating interactions with their AAC devices and paraeducators were rarely observed offering opportunities to initiate or providing support to help the student initiate.

Following training, data from each paraeducator indicated an increased rate when providing the communication intervention, and as a result, the student showed an increase in the targeted intervention, initiation.

The results of the intervention highlight the need for explicit teaching when promoting initiation among students with autism who use AAC.

The study by Yalim et al. [54], brings to mind the meltdowns of autistic children, in which they express their emotions through prolonged or loud sobs and screams, even to the point of self-harm.

Parents of autistic children have faced many implications and difficulties as a

result of this.

Children with autism encounter it when they are in a room or place where they meet many people, whenever they hear a loud noise, or when they are in circumstances where it is difficult for them to express their feelings or emotions. This study found an obsession in autistic children that could trigger meltdowns, and it was hard to change because of the close way the obsessions and autistic children's daily routines are linked.

The school must prepare preschool teachers and exceptional education teachers who will instruct special needs children and children with autism, particularly strategies for handling meltdowns, to support parents.

According to the Barker et al. [4] article, peer use of AAC as additional input led to a huge profit in speaking skills, but teacher use of AAC during instructing was associated with less positive growth.

The lack of training reported by teachers highlights the need for further research to promote the use of AAC in preschool, including facilitating interactions between AAC users.

The study by Tosun et al. [51] aimed to determine the preferences of individuals with ASD, determine how the preference assessment sessions were conducted, and which AAC system the participants preferred to use among the commonly used AAC systems:

- sign language;
- speech-generating device;
- picture exchange-based systems;

Most of the participants with autism spectrum disorder preferred speech-generating devices compared to sign language and picture systems.

In addition, most of the participants tended to prefer the AAC systems they learned the fastest.

The findings show that people with ASD prefer to use speech generating devices when requesting an object or action.

The incidence of autism is increasing dangerously rapidly, but diagnosing autism still requires a lot of time and a lot of money.

Early autism diagnosis might be very helpful for treating patients with appropriate medical care at the appropriate time.

The advancement of machine learning (ML) algorithms and AI has made it possible to identify autism quite early.

Although numerous research studies have been conducted using different techniques, they have not led to definitive conclusions about the ability to predict the characteristics of autism in different age groups.

Symptoms of ASD have been identified in one out of the 60 children in the US, based on an investigation released by the Organization for Prevention and Control of Disease (CDC).

Autism is believed to affect 2.65% of all school-age children in the nation-state of Korea in the survey carried out by Ravishankar et al. (2023) [48].

Figure 1.1 illustrates how ASD corresponds to a category of neurological disorders characterized by a lack of interpersonal connection that continues throughout a person's life in addition to daily and restrictive activities.

Investigations employing neurological approaches, such as Positron emission

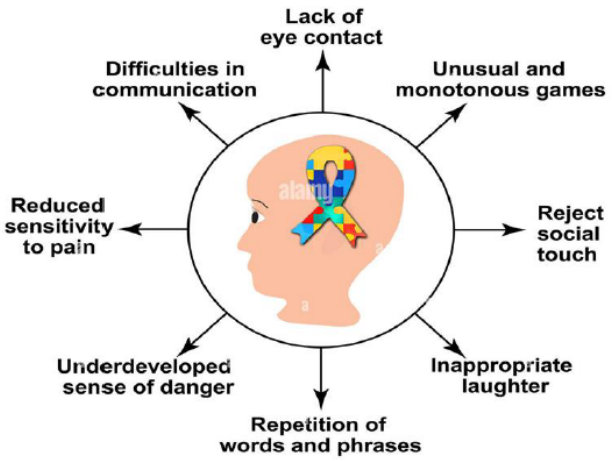


Figure 1.1: ASD [48] is related to a group of neurological conditions whose symptoms include a lack of social connection that lasts their entire lives as well as monotonous, constrained activities.

tomography (PET) or the use of magnetic resonance imaging (MRI), have provided an enormous amount of illumination onto the neurological factors behind ASD.

The use of ML approaches is currently being extensively used to classify individuals suffering from Autism and typical control (TC), to analyze biological facts using MRI information, and to predict the likelihood of the disease.

A small sample comprising less than 250 samples from previous ML examinations obtained an impressively high precision in the classification of 65%–96.27%.

The use of support vector machines (SVM), logistic regression (LR), randomly generated forests, linear discriminant analysis (RDA), as well as deep neural networks, are among the classification approaches employed in the associated examinations.

In order to improve the classification accuracy of patients with autism based on the full Autism Brain Imaging Data Exchange dataset in the work of Wang, C., et al. (2019) [52], the process consists of three basic steps:

- The resting-state functional magnetic resonance imaging data to calculate the functional connectivity (FC) based on the automated anatomical; labeling atlas with 116 brain regions.
- the SVM-recursive feature elimination algorithm to select the top 1000 features from the primitive FC features.
- trained a stacked; sparse auto-encoder with two hidden layers to extract the high-level latent and complicated features from the 1000 features;

Finally, the optimal features obtained were fed into the softmax classifier.

Autism is a category of early developmental deficiency that's challenging to detect, especially in young children, due to its symptoms being contingent on children's cognitive responses.

Failure to recognize and address autism between ages 2-5 makes treatment in

later stages more complex.

Many parents consider it challenging to describe any of their kid's symptoms. The faster diagnosis of autism is when a child presents only an earlier symptom as in Table 1.2.

The prevalence of ASD has significantly risen in recent decades, suggesting

Behaviors	a. Narrow interest b. Short attention span c. Intense tantrums
Play	a. Prefers solitary or ritualistic play b. Doesn't imitate action of others
Sensory	a. Rubs surface or Licks objects b. Withdraw from physical contact c. May find normal noises painful
Social	a. Displays lack of empathy b. Avoid eye contact c. Does not play interactive games
Communication	a. Repeats words b. Communicates with gestures instead of words c. Cannot start or maintain conversation

Figure 1.2: Table of significant signs of ASD.

an autism epidemic as clarified in the work of Chiarotti et al. (2020) [6].

Monitoring ASD helps estimate its occurrence and explore variations over time and regions. Prevalence data are gathered globally from health and education databases or specific studies.

A review of ASD prevalence estimates since 2014 reveals considerable global variability due to methodological differences in case detection.

Despite this variability, there's a consistent increase in prevalence within each region, particularly around the 2010s. Data sources impact prevalence estimates, with teacher/parent reports leading to higher estimates.

The population-based and administrative data methods each have advantages and disadvantages. Geographical area and study source contribute to prevalence differences. Environmental factors and genetic risks also influence ASD prevalence.

The article by Studer et al. (2017) [46] addresses the challenges in implementing early intensive behavioral intervention for autism in Switzerland, highlighting deficiencies in acceptance and government support.

It highlights a major gap between the US and most European countries and it

wants to be a Swiss pilot project.

The authors share their experience from an intervention center, while also noting obstacles within schools and the education system. The article underscores the need for greater efforts in Early Intensive Behavioral Intervention (EIBI), recommending additional funding, improved training, and consistent guidelines.

EIBI is a type of intensive behavioral therapy used to treat children with ASD in the early stages of their life, typically during infancy or early preschool age. Therapy's primary goal is to promote the development and improvement of social, communicative, and behavioral skills in children with autism. EIBI also involves the active involvement of parents and caregivers in the child's learning process, enabling strategies to be applied outside of therapy sessions.

The current diagnostic criteria for the autism spectrum in DSM-5 and the standardized diagnostic tools for autism lead to significant clinical heterogeneity and uncertainty, which could hinder fundamental research on autism mechanisms.

To improve clinical specificity and refocus research on authentic autism presentations, new diagnostic criteria are needed for prototypical autism between ages 2 and 5.

The research work by Mottron et al. (2023) [34] includes autism within other non-dominant, familiarly aggregated phenomena that share asymmetrical developmental bifurcations, such as twin pregnancy, left-handedness, and breech presentation/delivery.

According to this model, the nature, trajectory, and structure of positive/negative signs of autism would arise from the polarized question of whether language and information processing occur in a socially distorted manner.

Prototypical autism would follow a canonical developmental trajectory in which a gradual decline in social bias in processing incoming information, evident at the end of the first year, bifurcates into a prototypical autistic presentation in the second half of the second year of life.

This bifurcation event is followed by a plateau during which these atypicalities exhibit maximal stringency and distinctiveness, and eventually, in most cases, partial normalization.
During the plateau period, the orientation toward information processing is significantly altered, with an absence of bias for social information, contrasting with a high interest in complex, unbiased information, regardless of its social or non-social nature.
Integrating autism into asymmetrical developmental bifurcations would account for the absence of harmful neurological and genetic markers and the presence of familial transmission in authentic autism presentations.

From a technological standpoint, people with CCN fundamentally regard AAC boards as assistive technology tools.
CCN individuals are those who have trouble communicating and expressing themselves, such as those who have intellectual disabilities, cerebral palsy, aphasia, or autism spectrum disorder.
They are who:

- experience significant fatigue speaking;
- requires spelling or grammar support;
- communicate at very slow rates;

The AAC is a way of expression different from spoken language, that aims at increasing (augmentative) and/or compensating (alternative) the difficulties of communication and language of many people with disabilities.
AAC does not conflict with the rehabilitation of normal speech; instead, it enhances it.
In addition, it could contribute to its success wherever possible. Therefore, you should not wait to introduce it while a child is young, as soon as you see problems with spoken language development, or as soon as an illness or accident has damaged it.

There is no evidence that the use of AAC impairs or interferes with speech development or recovery.

Artificial sound communicators, personal computers, and tablets with specialized software allowing different kinds of access suited for individuals with severely limited mobility are instances of support items for communication. Additionally, they make it easier to include various pictographic and orthographic sign systems as well as other outgoing methods, including voice output. They might also employ conventional resources such as books and message boards.

The system of symbols for ACC is divided into gestural and graphic.

For those who are illiterate due to age or disability, pictographic methods are applied. They offer the advantage of allowing communication to progress from a very basic degree, adaptable for those with low cognitive abilities or in the very early stages, to a very rich and advanced level. However, they will never be as full and flexible as the level that can be achieved with the use of written language. Pictographic System of Communication (PSC) and ARASAAC are the two most popular pictographic systems in which sentences typically would include a sequence of standardized pictures.[1]

Support products for communication can be categorized into basic and technological types. Communication boards, which include surfaces with various materials holding graphic symbols (such as photographs, pictograms, letters, words, and sentences), serve as basic support products.

When symbols are distributed across different pages, they are referred to as communication books.

The use of support systems is effective when combined with additional factors, such as appropriate educational and therapeutic environments that enable the subject to be immersed in a supportive language environment, surrounded by sensitive and competent interlocutors, and engaged in meaningful and enriching activities.

[1] `https://arasaac.org`

Consequently, the most effective technique is the predictive task. The tool helps them to communicate by selecting and arranging pictograms in sequence to make up a sentence.
Such boards have enabled CCN adults and children to communicate and participate in a wide range of environments and activities denoting the same barriers.
The boards show pictograms divided into categories as “persons”, “objects”, “food”, and “adjectives”.
The AAC board is made of a content area with a selectable pictogram and a phrase area where the selected pictograms of the sentence are as in Figure 1.3.
The user may open many category folders or pass many pages to find a specific pictogram while pictograms near the needed one can act as distractors in searching activities.

Several investigations have been done in this area on Language Model (LM), including from statistical to neural network models.
The foundational work of this thesis is about the use of a BERT transformed-based model, introduced by Devlin et al. (2018) [8], to tackle the challenge of ”predicting the next pictogram” as described in the research paper by Pereira et al. [40].
Our contributions help move from single-modal AI approaches to multi-modal AI which maximizes their benefits.
The state-of-the-art model within the Multimodal AI area is analyzed and examined in the next chapter, which culminates with an in-depth description of our starting point solution.
The key to reducing the difficulties and simplifying the use of these devices is to find a strategy to ensure that the user will find the required pictogram quickly.

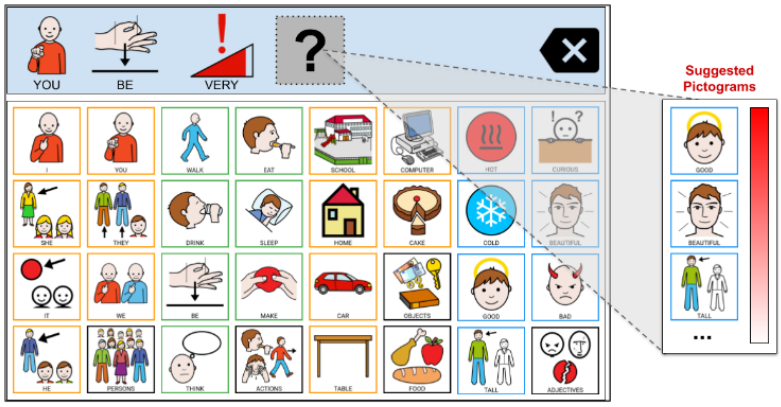


Figure 1.3: The AAC system [40]: a content area (large rectangle at the bottom) with the selectable pictograms and a phrase area (blue rectangle at the top) that presents the selected pictograms to form the sentence.

Chapter 2

State of the Art

Multi-mode AI represents a pivotal advancement in the field, seamlessly blending diverse data sources to amplify our capacity for comprehension and prediction. This innovative approach effectively surmounts the constraints associated with single-mode AI systems, heralding a new era of possibilities.

The strength of multi-mode AI becomes apparent in its unique ability to facilitate a level of detailed perception akin to human cognition. This remarkable feat is achieved through its adaptability to data presented in various formats, allowing for nuanced and human-like understanding.

The journey towards achieving multi-mode AI excellence traces a path of evolution in the AI landscape. It encompasses a transformative progression that begins with the advent of the LM and neural networks, culminating in the powerful Transformer architecture. This convergence of Natural Language Processing (NLP) and Computer Vision (CV) represents the pinnacle of this evolutionary trajectory, propelling AI capabilities to unprecedented heights.

At the heart of this transformation lies the pivotal role of multimodal models. These models, shaped by the flexibility of data in diverse formats, exhibit a remarkable capacity to adapt to a wide array of tasks. Their versatility makes

them indispensable in harnessing the full potential of multi-mode AI.

The evolution from Statistical LM to advanced neural models, exemplified by the likes of BERT, reaches its apex with the emergence of Large LMs (LLMs). These formidable LLMs not only enhance our ability to interpret data but also significantly bolster the reasoning capabilities of AI models.

2.1 Open Challenges

Transitioning into the realm of multimodal AI brings forth a set of intriguing challenges that demand innovative solutions and research efforts.
Challenges include explainability and interpretability in multi-modal models, aligning text and images without labels, and processing multiple input modalities together to achieve better and more comprehensive results.

- **Interpretability and Explainability**: Explainability and interpretability are critical to understanding how Vision-and-Language (VL) models combine textual information.
  The Python library Xplique [12] provides a demonstration of this necessity by improving the explainability of deep learning (DL) models via techniques like gradient-weighted class activation mapping.

- **Alignment with Unlabeled Attention**: Models learn to link text words to visual regions without human annotations, using multi-task pretraining techniques, including the new Masked Visual-token Modeling (MVM) task [13] that retrieves visual tokens from masked frames to learn text-image correlation.

- **Multimodal Models** Another crucial aspect is the construction of end-to-end learning pipelines that integrate LLMs with other modules (e.g. graph Neural Network (GNN), custom layers, and external memories)

to improve the understanding and representation of multimodal data.
A large-scale memory encodes multimodal world knowledge via a unified encoder in ReVEAL to deliver effective advanced information retrieval results.
Other examples of architectures are BERT-like transformers integrated with additional layers to process multimodal inputs combined with deep metric learning to create latent space where inputs are separately embedded [33].

Exploring another direction, Graph modeling, still little explored for visual data, could be used to incorporate commonsense and domain facts from external knowledge bases and to train networks capable of working directly on structured and unambiguous semantic representations. The introduction of Vision GNN architecture by Han et al. [15], to extract graph-level features for visual tasks, revolutionizes image processing.

- **Memory consumption** Managing computational resources efficiently is a paramount concern in the realm of DL, and the study by Raschka et al. [27] stands as a testament to the ongoing endeavors to address memory consumption issues while preserving model effectiveness and precision.

2.2 LLM and Transformer: Shaping the Future of AI Models

In the ever-evolving landscape of AI models, it's crucial to examine how LLMs and Transformers have come to redefine the field. The rise of LLM models, exemplified by the popularity of the pre-trained Transformer-based GPT-4 [36], signifies a significant milestone. These models are not limited

to NLP but extend their reach into diverse domains, including CV and graph learning. In CV, Vision Transformer (ViT) plays a pivotal role, transforming images into a series of patches akin to word embeddings, while Graph Transformer Networks are reshaping graph-related tasks without the need for domain-specific knowledge.

Furthermore, the survey conducted by Zhou et al. (2023) [59] underscores the extensive study of Pretrained LMs across the three major AI fields: NLP, CV, and GL. Concurrently, we mustn't overlook the significance of bidirectional encoders like BERT, which predict masked words and contextual sentences, albeit with certain limitations. On the other hand, autoregressive decoders, exemplified by GPT, excel in text generation but may lack the ability to capture bidirectional interactions effectively. This rapid evolution of models necessitates refined toolkits like LLaMA2-Accessory [56], designed to facilitate pre-training, fine-tuning, and implementation of LLMs. Additionally, exploring the dynamic field of CV task pre-training through techniques like Auto-supervised learning (SSL) brings us closer to achieving more robust models, in Figure 2.1, thanks to the transferability of representations learned during pretraining to subsequent supervised tasks.

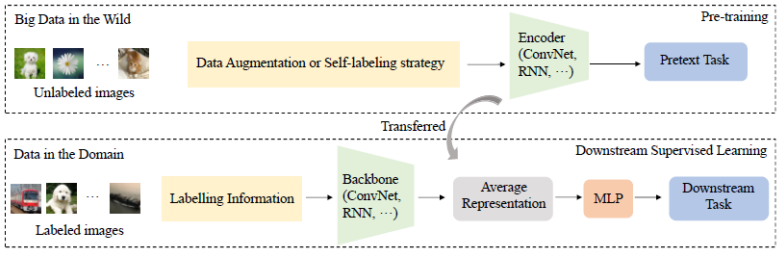


Figure 2.1: The general pipeline for SSL. The top part represents the pretraining, and the bottom stream obtains transferred parameters from above to learn downstream supervised tasks.

2.3 Multimodal Large Language Model

The rise in popularity of the Multimodal Large Language Model (MLLM) marks a significant shift in the field of AI. This transformation is exemplified by the evolution of chatGPT into a multimodal entity with the advent of IDEFICS [23]. It's the first open ChatGPT-style multimodal model based on Flamingo, seamlessly integrating both images and text to generate conversational outputs.

The core idea behind this multimodal approach is to enhance model interpretability. By fostering interaction between visual and textual modalities, MLLMs facilitate the creation of meaningful semantic links between linguistic tokens and the visual elements within images. This, in turn, contributes to a deeper understanding of complex models, enhancing their reliability in critical domains such as medical care and autonomous driving.

Several studies have delved into the abstract reasoning capabilities of multimodal models. Notably, KILOGRAM [19] underscores the significance of fine-tuning and detailed dataset descriptions. These elements favor the establishment of semantic connections between different parts of images and words. In their work, Ji et al. introduced a visually annotated tangram dataset aimed at assessing the abstract reasoning capabilities of multimodal models. This study demonstrates that the integration of visual and linguistic representations significantly enhances abstract reasoning abilities, unlocking new possibilities in the analysis of complex data. It further emphasizes the pivotal role of the fine-tuning phase in optimizing multimodal models.

The comprehensive MLLM survey conducted by Yin et al. in 2023 [55] further explores the integration of LLM with the visual world. This survey

encompasses various model types and approaches, including Multimodal Instruction Tuning (M-IT), Multimodal In-Context Learning (M-ICL), and Multimodal Chain-of-Thought (M-CoT). These approaches offer fresh perspectives on human-software interaction and open doors to novel applications and capabilities in the realm of AI, including enhanced reasoning abilities.

2.3.1 Multimodal CoT

LLMs have demonstrated impressive complex reasoning abilities when handling multimodal input converted to text. However, this transformation can result in a loss of information, which necessitates a closer examination of their functioning within multimodal models.

LLMs serve as the foundation for cutting-edge solutions, serving as the primary processing unit in multimodal models. Notably, LLMs have excelled in complex reasoning tasks by generating intermediate reasoning steps before arriving at a final answer, popularizing the CoT reasoning technique.

The M-CoT paradigm represents a significant advancement in reasoning approaches, as it involves multimodal input and breaks down complex tasks into a series of intermediate reasoning steps. However, it's worth noting that earlier LLMs with fewer than 100 billion parameters sometimes produced hallucinatory results, leading to errors in answer deduction.

To address this issue, a transition to multimodality was adopted, wherein the inference of the answer benefits from better reasoning generated based on multimodal information. MLLMs have effectively mitigated this problem within the M-CoT framework. These MLLMs produce output that combines rationale with input, providing models with limited parameters the capability to yield meaningful results. This approach leverages visual features to reduce

the risk of erroneous conclusions.

An illustrative example of M-CoT reasoning is presented in the article by Zhang et al. [57]. The authors introduce a framework known as M-CoT Reasoning, which leverages the synergy between language and vision. They employ diverse learning strategies that outperform models like GPT-3.5 and even human performance on reasoning tasks.

Another noteworthy contribution to M-CoT prompting is introduced by Huang et al. in their work involving Kosmos-1 [18]. This approach harnesses intermodal transfer to enhance performance in perception-language tasks. The process involves two stages: initially, the prompt takes an image and guides the model to generate a rationale. Subsequently, the model is provided with the rationale and a task-aware prompt to produce the final results. This approach, as exemplified in Figure 2.2, significantly contributes to the accuracy of tasks like Visual-QA by ensuring that the model accurately describes the content of the image in text to predict the answer.

2.3.2 Retrieval-Enhanced Multimodal Language Models

Speaking of a learning pipeline based on LLM integrated with modules like external memory, we refer to the model mentioned below. A new paradigm in knowledge-based information retrieval is ReVEAL from the article by Hu et al. [17]. This multimodal model leverages a combination of memory, encoder, retriever, and generator to deliver effective advanced information retrieval results. The effectiveness of multimodal architectures is obtained through the pre-training and fine-tuning phases.

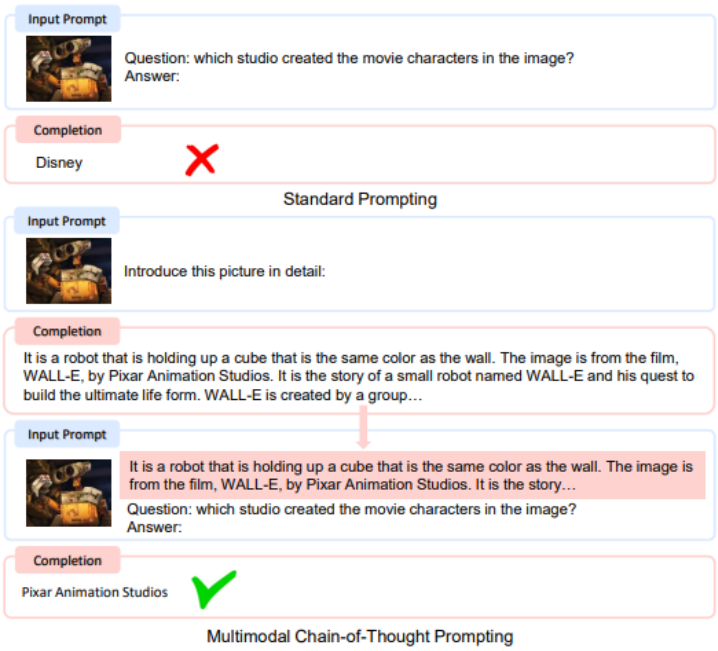


Figure 2.2: Kosmos-1 [18] can tackle challenging question-answering and reasoning problems by first generating a rationale thanks to M-CoT prompting.

2.3.3 Video Modality

In the realm of multimodal models, as explored in the article by Fu et al. [13], our focus now shifts to the dynamic interplay between video and text inputs. The VIOLET model demonstrates how the link between text and image is learned through the pre-training phase on masking tasks for both text and images, as illustrated in Figure 2.3. It represents a major step forward in understanding temporal dynamics in video through the use of transformers. Based on the Video Swin Transformer, it extracts spatial and temporal representations of images across patches, then uses a Language Embedder and a Cross-Modal Transformer to merge the multimodal information.

2.3.4 One model for all modalities

Emu, cited in Sun et al. [47] represents a major advance in multimodal computing through the adoption of a unified architecture leveraging LLMs.

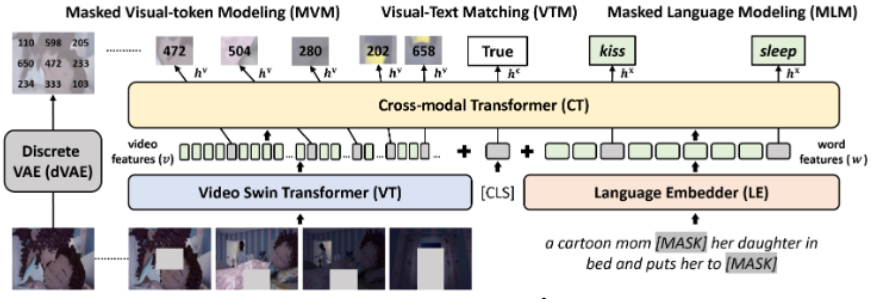


Figure 2.3: VIOLET [13] components include VT, LE, and CT, using a discrete VAE to extract discrete visual tokens in order to implement different strategies during large-scale text and image pretraining.

In order to anticipate the following visual or textual token in an autoregressive fashion, this model learns from videos and images intermingled with text. Interspersing images with text gives an intuitive representation of the complex concept, improving the ability for multimodal learning and anchoring to real-world concepts.
On a broader front, UnIVAL, introduced by Shukor et al. [45], tackles the challenge of unifying text, audio, and video modes within a unified model, emphasizing the efficiency of pre-training on diverse multimodal tasks. It mixes text, graphics, video, and audio into one structure with 0.25 billion parameters. UnIVAL is efficiently pre-trained on a wide range of multimodal tasks while other work shows that multimodal models can learn text-image associations without the pretraining step.

In contrast, TVLT by Tang et al. [49] represents a significant advance in multimodal learning without the use of text. This end-to-end VL transformer operates on low-level visual cues and audio data to learn multimodal representations. Avoiding text-based modules like tokenization or automatic speech recognition, it learns from the reconstruction of masked visual patches or audio spectrograms. Compared to text-based models, TVLT is significantly faster at inferences and requires fewer parameters, demonstrating its ability to

learn visual-linguistic representations from audiovisual inputs without the explicit use of text.

Yet, Schwettmann et al. from 2023 [43] reveal a surprising twist in the narrative. They show how LMs, originally intended for textual data, can expand their expertise into the visual domain through a transferable learning mechanism, illustrating the adaptability of LLMs to multimodal contexts. Surprisingly, LMs demonstrate the ability to extend the representations learned in one modality to other modalities. A transformer-based LM, originally intended for text, can acquire visual knowledge through a transferable learning mechanism. This involves a self-supervised visual encoder and a linear projection trained on image-to-text translation tasks.

Moving into the realm of CV, the framework shown in Figure 2.4, LENS by Berrios et al. 2023 [5], is a modular approach to CV problems leveraging the power of LLMs. The LLM reasons on the output of a set of visual modules that return textual information about the image. It's a modular solution integrating language and vision skills. The visual module extracts textual information using pre-trained visual models, text is the input to the LLM enabling tasks such as object detection. The strength of LENS lies in eliminating the gap between the different modalities at zero cost since it does not require multimodal pretraining phases or extra data. The model works "cross-domain" without additional cross-domain pretraining phases satisfying CV tasks by exploiting the abilities of LMs that operate on descriptions of visual inputs without further vision-language alignments.

Lastly, the recently published work of Lei et al. (2023) [24] presents ViT-LENS which extends the capabilities of a pre-trained ViT to perceive and comprehend diverse modalities beyond 2D images. It employs Modality Embedding and the Perceiver architecture to map modality-specific data into the pre-trained ViT input space. Then the encoded output of ViT is aligned with the

feature extracted from the data's anchor text/image/text-image, through an off-the-shelf foundation model. This novel approach enables a pretrained ViT to integrate and understand diverse modalities beyond images while leveraging its knowledge from the pretraining to better comprehend and interpret these modalities. The 3D Shape Encoder in VIT-LENS includes a point embedding layer, a Perceiver, and a pretrained CLIP-ViT model. To ensure optimal encoding of 3D point clouds, heuristic designs are employed: partitioning input point clouds into patches and mapping them into point embeddings, followed by using the Perceiver to distill these embeddings into latent embeddings for input to CLIP-ViT. The Perceiver architecture plays a crucial role, iteratively distilling inputs into a latent bottleneck, making it versatile for handling various modalities. ViT-LENS uses the Perceiver to map input signals from different modalities into CLIP-ViT's input space. For representation alignment, a contrastive loss function is applied to triplets of 3D point clouds, rendered images, and associated text descriptions enhancing representation learning for 3D data and enabling omni-modal capabilities efficiently as in Figure 2.5.

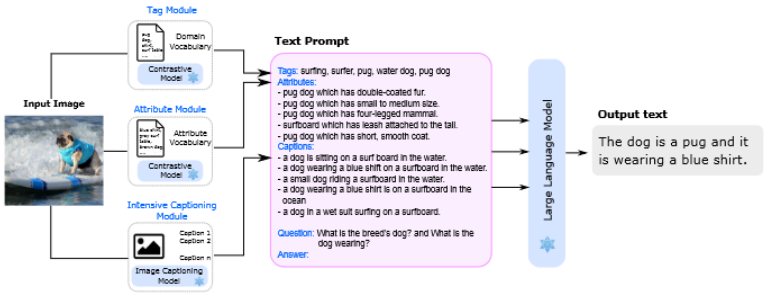


Figure 2.4: In the [5] LENS framework, visual modules retrieve a textual description for an image, used by the LLM reasoning module to generate an answer to a query.

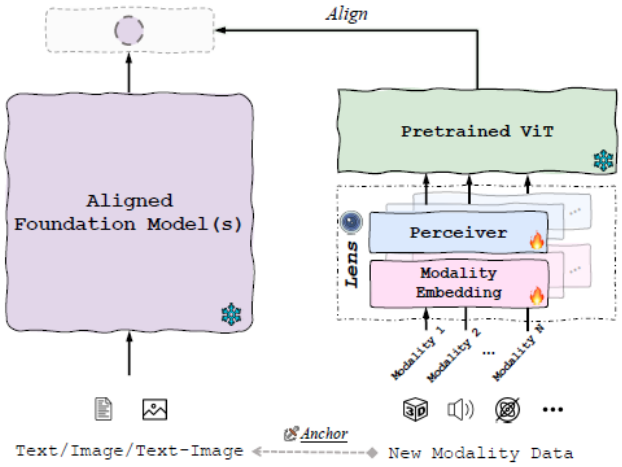


Figure 2.5: ViT-LENS architecture [24]: the capabilities of the pretrained ViT are extended by Modality Embedding and the Perceiver architecture. The encoded output of ViT is aligned with the feature extracted from the data's anchor text/image/text-image, through the foundation model.

2.4 Applications of Multimodal Models Across Various Contexts

In today's world, multimodal models play a crucial role in various applications, ranging from robotic coordination to medical analysis. In this section, we will explore some of these key applications.

2.4.1 Applications in Robotic Coordination and Human-Robot Interaction

Multimodal models have found a wide range of applications, including robotic coordination, cascade agents, and robotic industrial field guiding systems. These models exhibit a remarkable ability to collaborate with both real and virtual robots, highlighting their versatility in human-robot interaction scenarios.

A notable contribution in this domain is the work of Driess et al. [10], where the MLLM PaLM-E introduced to coordinate real and virtual robots is presented. This innovative model is designed to facilitate coordination between

real and virtual robots and is trained on a combination of vision-language tasks. By demonstrating its capability to transfer knowledge from visual language domains to decision-making processes, PaLM-E empowers robots to efficiently perform planning tasks. It's a decoder-only LLM that generates textual completions autoregressively given a prefix or prompt based on PaLM as the pre-trained LM but Embodied. The main architectural idea of PaLM-E is to inject continuous, embodied observations such as images, state estimates, or other sensor modalities into the language embedding space of a pre-trained LM.
It is trained in task VL, demonstrating skills in sequential reasoning and mathematical operations, and transfers knowledge from planning robots to answering questions.

2.4.2 Medical Applications of Multimodal Models

In this subsection, we delve into the application of multimodal models in the medical field, specifically focusing on the analysis of medical images and visual conversations within the medical-surgical context.

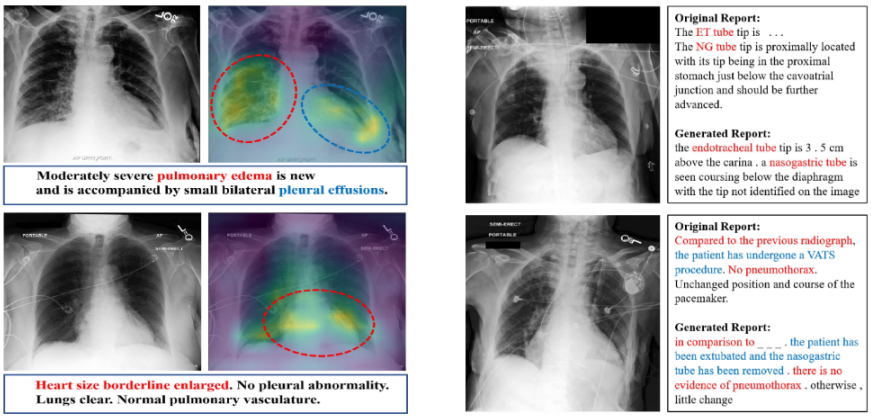
Moon et al. [32] have undertaken extensive research in multimodal representational learning tasks within the medical domain, utilizing radiological images and unstructured relationships.

One notable outcome of their work is the introduction of MedViLL, a BERT-based model featuring innovative multimodal attention masking scheme. MedViLL has showcased its superior performance across four crucial subtasks: diagnosis classification, medical report-image retrieval, answering medical visual questions, and generating radiological reports (see Figure 2.6).

XrayGPT, a conversational medical model developed by Thawakar et al. (2023) [50], takes a unique approach by combining vision and language modalities to analyze and respond to inquiries regarding chest radiographs (Figure 2.7).
This model aligns visual features and textual information by leveraging a pretrained medical vision encoder called Med-Clip and a medical LLM based on Vicuna. By incorporating interactive summaries from radiology reports, XrayGPT demonstrates exceptional conversational capabilities, underpinned by a deep understanding of chest radiographs. This enhances the performance of LLM in the medical context.

In an August 2023 article by Google Health AI [7], authored by Corrado and Matias, the implications of multimodality in medical AI are explored. The article showcases various approaches to incorporating multimodality into LLMs, highlighting the potential of multimodal medical LLMs to integrate diverse data modalities and enhance diagnostic and analytical capabilities in medical settings.

Furthermore, the article underscores the significance of Med-PaLM, a LM developed by Google. Med-PaLM effectively harnesses structured knowledge and clinical multimedia content, such as that from the United States Medical Licensing Examination (USMLE), to enhance the accuracy of medical question answering. In particular, Med-PaLM 2 has demonstrated a notable increase in accuracy compared to its previous version when tackling USMLE-style questions.



(a) Attention map visualization. (b) Radiology report generation analysis.

Figure 2.6: MedViLL [32]: the left image highlights the extracted attention regions (a), the right image compares the generated report and the original report on the same chest radiograph image (b).

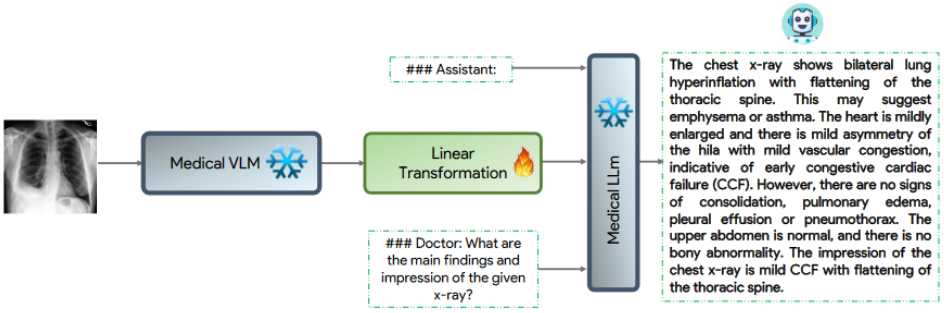


Figure 2.7: XrayGPT [50] architecture in which a radiographic image is processed through three sequential components: 1. Frozen medical visual encoder to extract diagnostic information, 2. Adaptive linear transform layer to align visual features with the LLM, and 3. Medical LLM to generate a detailed summary based on the features and prompt provided.

2.5 PictoBERT: Bridging the Gap in AAC Communication

AAC has seen remarkable advancements in recent years, with predictive models playing a crucial role in transforming the landscape. These models offer innovative solutions to assist individuals with CCN in constructing sentences with pictograms, thereby enhancing their communication efficiency. In this section, we delve into the state of the art in AAC predictive models and then

introduce PictoBERT, a powerful Transformer-based model designed to improve the efficiency of this predictive task.

2.5.1 State of the Art in AAC Predictive Models

The primary focus of this debate is to delve into the current state of the AAC field, moving away from a broader perspective.

Predictive models play a pivotal role in AAC, offering a solution to the challenge of selecting the next pictogram based on the previous one. These predictive methods hold significant potential to assist AAC users in various ways:

- **Sentence Construction**: Predictive models reduce the number of selections required to construct a sentence, enhancing communication efficiency;
- **Spelling Support**: They provide spelling support for individuals who struggle with spelling words;
- **Grammatical Assistance**: Predictive models offer grammatical support, particularly beneficial for those with morphosyntax difficulties, promoting coherent communication;
- **Enhanced Communication Rate**: By suggesting the next pictogram, these models accelerate the communication rate, making interactions more fluid.

In the realm of pictogram prediction in AAC, there have been notable developments, primarily focusing on semantic grammar [31] and knowledge graphs [41] as the foundational approaches. These approaches leverage the semantic relationships between words, incorporating ontology-based knowledge and semantic roles to make predictions. However, it's essential to note that these mechanisms are not purely probabilistic. They return pictogram sets without specifying which is the most appropriate [39]. Key points to highlight include:

- **Lexical-Semantic Hierarchy**: These approaches prioritize lexical-semantic knowledge hierarchy, reducing ambiguity compared to relying solely on grammatical classes;
- **Static Grammar Structure**: The use of static grammar structures can necessitate processing the entire construction pipeline when changing vocabulary, highlighting potential limitations.

One approach involves completing the grammatical structure of a sentence, often starting from a semantic role such as the verb. For example, the Colourful Semantics' structure sequence involves Agent, Verb, Theme, Location, Recipient, Manner, and Time. This structured approach can resemble a fill-in-the-blank task, enhancing predictability.

Statistical LMs, incorporating unigrams and bigram models containing word usage knowledge, have demonstrated improved results for communication systems [35]. These models are applied to pictograms, utilizing a corpus of sentences related to various locations as a training dataset for constructed LMs to make location-aware pictogram predictions [14].

Prediction granularity can vary, focusing on the pictogram itself or its part-of-speech (POS) tag in the n-gram statistical LM [16]. This granular approach improves the immediate availability of desired pictograms.

Furthermore, word-sense language modeling is introduced, particularly relevant for Word-Sense Disambiguation tasks. For instance, SenseBERT [26] combines word and super-sense embeddings to predict masked words and their super-senses. This approach, however, introduces some ambiguity due to the nature of super-senses. Super-sense is a WordNet cluster of senses and may be useful to suggest classes of pictograms but is very ambiguous.

To address the multi-sense language modeling challenge, Gated Recurrent Units and Graph Attention Networks [25] are employed at the granularity of

WordNet word senses. WordNet uses the granularity of word senses to generate two probability distributions at each place in the corpus, one covering the vocabulary of words and one covering the vocabulary of senses, both of which achieve an accuracy greater than 22%.
The prediction is based on the previous words and not on earlier word senses, so there can be ambiguity when transforming pictograms, that are better represented by a word sense, into a word.

2.5.2 PictoBERT: Transformers for next pictogram prediction

The foundation of our exploration into PictoBERT begins with the groundbreaking work of Pereira et al. [40]. Their research offers a compelling solution grounded in the BERT transformer model, aimed at addressing the intricate challenge of predicting the next pictogram in a sentence under construction. In this context, pictograms represent graphical depictions of concepts, each adorned with labels denoting actions, objects, persons, animals, or places. The primary goal is to enhance the performance of AAC boards, which facilitate communication for individuals with CCN.

PictoBERT revolves around two pivotal concepts: word-sense usage and modified BERT input embeddings. These concepts underpin its capacity to outperform traditional n-gram models. The paramount feature contributing to its success is transfer learning, which endows PictoBERT with remarkable flexibility for fine-tuning to meet diverse application demands.

As previously discussed, the landscape of predictive models in AAC presents various approaches, including statistical LMs based on n-grams, semantic grammar knowledge bases utilizing fill-in-the-blank techniques, and transformer-based models like BERT. PictoBERT stands out due to its ability to provide

contextualized predictions, a feature that empowers it to consider the entire sentence's context when predicting the next pictogram, distinguishing it from other solutions that lack this contextual depth.

PictoBERT leverages the power of WordNet word senses, which serve as the foundation for constructing a corpus dedicated to representing pictograms. In this context, the refined task is predicting the next word sense. A key architectural innovation involves modifying the BERT embedding layer to refresh its vocabulary using ARES, which facilitates the transition to a vocabulary rooted in word senses.

The corpus used is part of the Child Language Data Exchange System (CHILDES) corpus transformed into the word-sense corpus Semantic CHILDES (SemCHILDES). The model should be embedded into the AAC tool. The corpus is discussed in detail in section 3.2.1.

The assessment of PictoBERT's performance in comparison to n-gram models is based on intrinsic metrics, including perplexity (PPL), which measures the comprehension level of the LM, and the Top-N metric, evaluating how often the expected pictograms rank within the Top-N predicted outcomes. During the evaluation process, the PictoBERT vocabulary was customized to use only the pictograms present in the ARASAAC database. This customization is crucial to adapt the model to the specific needs of users. Furthermore, the model underwent a fine-tuning process using a portion of the SemCHILDES corpus. This additional training step contributed to achieving superior results in overall evaluation metrics.

ARASAAC is a valuable resource offering graphic and material assets under a Creative Commons license, designed to enhance communication and cognitive accessibility. This initiative garners support from the Department

of Culture, Sports, and Education of the Government of Aragon (Spain), with a specific focus on fostering innovation and professional training. The models are compared over ARASAAC pictograms.

Customization is a vital aspect of PictoBERT's applicability, catering to diverse user needs. The model's adaptability is exemplified through fine-tuning experiments, which showcase its ability to conform to varying settings, depending on user or group requirements.

In Figure 2.8, we provide visual examples of pictograms along with their corresponding word senses, illustrating the content-related descriptions of these graphical representations.

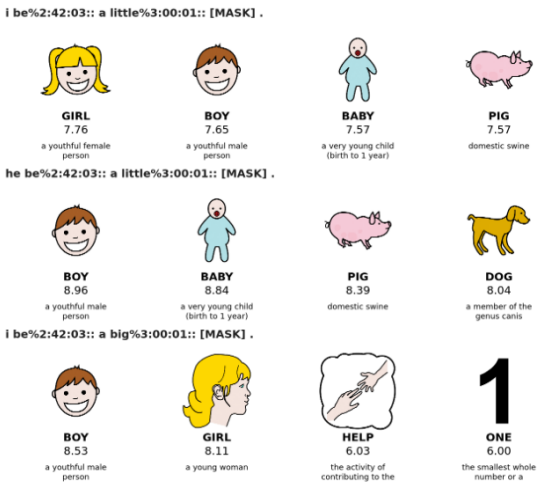


Figure 9: Pictogram prediction samples. Top of the figure: neutral gender sentence; center: male gender sentence; bottom of the figure: adjective changed sentence.

Figure 2.8: Pictogram prediction bu PictoBERT [40]: The top four pictograms for each sentence, along with their probability and WordNet definitions, demonstrate the potential for predicting pictograms in similar settings. When the gender is male or neutral, the models behave differently. The outcome is directly influenced by the training corpus and vocabulary employed.

In the following paragraphs, we delve into the internal workings of PictoBERT, shedding light on its main components and innovations.

A fundamental aspect of PictoBERT's design involves a significant overhaul of the BERT model's vocabulary and embedding layer. The first step in this process is the incorporation of WordNet word sense vocabulary, which fundamentally transforms BERT's original vocabulary from a word piece-based structure to a word sense-based one. This shift is pivotal for achieving a deeper understanding of language and context within the AAC domain. To facilitate this transition, a trained Word Level tokenizer, integrated with the SemCHILDES dataset, is employed. This tokenizer is responsible for splitting the tokens in a sentence based on whitespace and encoding the corpus, making it compatible with PictoBERT's input requirements. However, merely updating the vocabulary isn't sufficient. The embeddings layer, responsible for converting tokens into vector representations, must also be adapted to accommodate the new vocabulary. This crucial step ensures that PictoBERT comprehends the nuances of word senses and functional words.

Additionally, context-AwaRe Embeddings of Senses (ARES) is computed using BERT, ensuring that the embeddings exist in the same vector space. ARES embeddings serve two purposes: one set is computed from contextual examples, while the other is derived from glossary definitions. The integration of ARES embeddings replaces the original BERT embeddings, following the below guideline.

When a word sense from the vocabulary is identified, the vector corresponding to its sense key in ARES is obtained and inserted into the PictoBERT embedding layer. In the case of functional words, BERT's original embeddings remain in use. Word embeddings play a pivotal role in representing text in a numerical form, assigning a d-dimensional real-valued vector to each word in a text corpus. This vector essentially encodes the meaning of the word. The introduction of ARES by Scarlini et al. in 2020 provided a semi-supervised method for generating word sense embeddings in WordNet.

WordNet, developed by Miller et al. in 1995, is a lexical database that categorizes nouns, verbs, adjectives, and adverbs into synsets, which are groups of synonyms. Each synset represents a distinct concept, complete with a glossary definition and various lexical relationships, such as meronymy, hypernymy, and hyponymy. For each word sense present in WordNet, ARES identifies occurrences of these senses in a text corpus and computes their embeddings using BERT. This approach considers the entire context of a sentence to produce word embeddings. Furthermore, ARES computes embeddings for the glossary definitions of word senses by averaging BERT representations for words found in these definitions. The final representation is a 2048-dimensional vector, split into two aspects: contextualized (first 1024 positions), derived from usage examples, and gloss-based (last 1024 positions), computed from glossary definitions.

However, it is important to note that WordNet does not include word senses for functional words, such as pronouns, prepositions, conjunctions, and determiners. Moreover, BERT's special tokens, including [CLS], [MASK], [PAD], [UNK], and [SEP], are also considered.

To address the representation of multi-word expressions with a single pictogram, WordNet provides word senses for some popular expressions, such as "good morning" but not for all. In cases where ARES lacks embeddings for such expressions, PictoBERT utilizes BERT's tokenizer to tokenize the expression and subsequently computes its representation by averaging the embeddings from BERT. For example, the expression "I am fine" is tokenized into "I" "am" and "fine", and the embeddings of each token are extracted from BERT's embeddings layer. These embeddings are then aggregated into a mean vector, resulting in the expression's embedding representation.

To integrate these different embeddings, PictoBERT employs a vocabulary, denoted as V, comprising tokens representing word senses and functional

words. The embeddings in ARES, denoted as A, is of dimensionality $Rd \text{»} Dk$, where $d = 1024$ (considering one of ARES sides), and $Dk = 206,949$(the number of word senses in WordNet). Meanwhile, BERT's original embeddings, represented as B, have dimensions of $Rh \text{»} Dd$, where h is the hidden states' size and Dd is the BERT vocabulary size. Finally, a new embedding matrix, P, with dimensions $Rh \text{»} Dv$, where $Dv = |V|$ represents the PictoBERT vocabulary size. For each token ti in V, the matrix P is populated using A embeddings if ti represents a word sense, or B embeddings if it represents a functional word, following the algorithm outlined in Figure 2.10. In cases where ti is neither in A nor B, the ARES mean vector A is assigned to its position. This action accommodates customized word sense identifiers in the vocabulary, considering that AAC users may employ regional terms that might not be present in ARES.

Despite these advancements, there are inherent limitations in WordNet, such as the absence of word senses for functional words like pronouns, prepositions, conjunctions, and determiners. Additionally, some multi-word expressions may not have embedding representation in ARES.

To address these issues, PictoBERT employs BERT's tokenizer to tokenize such expressions and computes their representations by averaging the embeddings from BERT. This ensures that even regional or specialized terms can be accommodated within PictoBERT's vocabulary.

PictoBERT is pre-trained using the SemCHILDES dataset with specific hyperparameters. The training follows a similar pattern to BERT, with 15% of tokens selected for prediction, replaced with [MASK] tokens, random tokens, or left unchanged. During pre-training, if the ith token is selected, it is replaced with the [MASK] token 80% of the time, with a random token 10% of the time, and remains unchanged 10% of the time.

One of the remarkable features of PictoBERT is its adaptability. It can

be customized to suit the specific needs of users by training a new tokenizer and updating the embedding layer. This adaptability is essential for accommodating diverse user vocabularies and preferences. The mapping process transforms the input sequence of pictograms into a sequence of word senses, as PictoBERT operates on word senses as input. This mapping must be performed iteratively each time a user inserts a new pictogram into the sentence, with a [MASK] token added to the end of the constructed sentence. However, inherent ambiguities in language may persist.

For example, a pictogram representing ”lunch” may have multiple senses, potentially leading to confusion. In such cases, the AAC developer may design the system to ask the user for disambiguation, ensuring that the intended meaning is accurately conveyed.
Conversely, when transitioning from word sense to pictograms, some limitations may arise, resulting in the loss of certain pictograms in the user's vocabulary. To mitigate this, alternative methods, such as setting a probability threshold or selecting the top-k predictions, can be employed to enhance user experience.

The inference process in PictoBERT involves several key steps. First, the input sequence is tokenized, which transforms the word-senses and functional words into numerical representations based on their positions in the embeddings matrix. Additionally, numerical references to the [CLS] and [SEP] tokens are added to the sequence's beginning and end.

Next, PictoBERT processes the tokenized sentence and generates a vector representation for each token, including the masked token, using the attention mechanism. These vector representations are crucial for understanding the context and relationships between tokens. At the top classification layer,

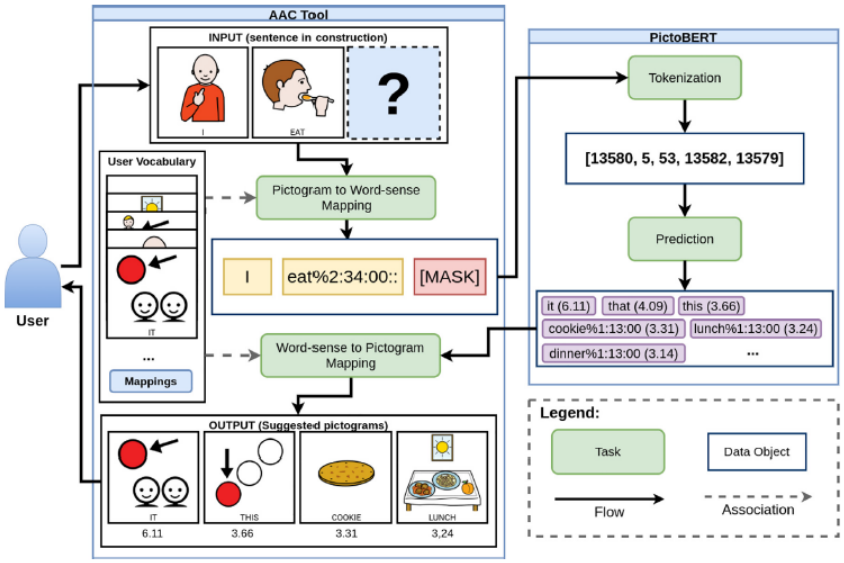


Figure 2.9: Workflow chart exploiting PictoBERT for predicting pictograms in AAC systems: the input is the user's selected sentence, the AAC maps the pictograms to word-senses and functional words to pass them as input to PictoBERT; the model tokenizes the sentence and makes predictions; the model's outputs are mapped to pictograms; the suggested pictograms are showed bask to the user.

Algorithm 1: BERT embeddings update

B = original BERT keyed Embeddings
A = ARES keyed embeddings
W = vocabulary from a corpus with words and word-senses
N = new keyed embeddings
foreach $w \in W$ **do**
  if $w \in B$ **then**
    $N_w = B_w$
  else
    if $w \in A$ **then**
      $N_w = B_w$
    else
      $N_w = \overline{B}$
    end
  end
end

Figure 2.10: The procedure for the embedding update is based on Algorithm 1 in the creation of PictoBERT models. It explains how ARES embedding replaces BERT embedding.

the model further processes the vector representations to approximate the vocabulary tokens. It generates a probability distribution across the vocabulary, which helps in predicting the next pictogram or word sense to complete the sentence. This process of predicting the next element is a fundamental aspect

of PictoBERT's functionality.

It's essential to acknowledge the limitations of the starting point of this research. These limitations are the following; The corpus used, derived from children's speech, is not inherently an AAC dataset. It may not fully represent the language used in AAC boards. To provide a real evaluation, further testing by AAC users is necessary to assess its practical utility. Implementing an entire AAC system that effectively integrates PictoBERT is a complex endeavor that extends beyond the scope of this work. PictoBERT is compared with statistical N-gram LMs, presenting a challenging comparison due to the unavailability of datasets from previous studies.

A statistical LM N-gram is a method used to predict words or tokens in a text based on the probabilities of word sequences. These models assess how frequently specific word sequences appear in the training data to make predictions. In essence, N-gram LMs provide a way to capture the likelihood of word combinations based on their observed frequencies in the training data. By considering different N values (e.g., 2-gram, 3-gram, and so on), these models adapt to varying levels of context and sequence length, thus improving their ability to generate accurate predictions within a given context.

However, increasing N also increases the model's complexity and demands more training data to cover all possible word sequences. Furthermore, raising N may result in larger and heavier models, which require more computational resources for both training and real-time applications.

This section outlines various directions for using PictoBERT in practical AAC applications. It emphasizes the significance of fine-tuning the model to accommodate the diverse needs of AAC users. Fine-tuning can be carried out with smaller datasets or even on tasks unrelated to pictogram prediction. For

instance, an AAC developer can collect the sentences constructed by a user in an AAC system and use them as training data for PictoBERT, tailoring the model to the specific user's communication style and preferences.

In conclusion, PictoBERT demonstrates its superiority over n-gram-based models in terms of linguistic comprehension, despite requiring more time and computational resources. The evaluation, conducted using various metrics, consistently showcases PictoBERT's exceptional performance.

Moreover, PictoBERT exhibits remarkable adaptability, making it a valuable tool for enhancing communication among individuals with CCN. Its flexibility extends to diverse user settings and activities, illustrating its potential to cater to the unique needs of the AAC user population. It can be fine-tuned with ease using various types of data, further underlining its versatility and applicability in real-world AAC scenarios.

Chapter 3

Proposed Solution

This thesis work is an extension of the mentioned research, introducing PictoViLT, a model that integrates textual and visual data as input to fine-tune a ViLT multimodal transformer. Through various masking strategies, my study evaluates the performance of PictoViLT, comparing it with PictoBERT and statistical n-grams LMs.
The prefixed goal is to surpass PictoBERT as a fundamental AAC tables communication methodology for CCN individuals.

The part of the code representative of how the concepts discussed in this Chapter were implemented can be found in the next Chapter 4, where illustrations of Python code detailing the structure of the main classes used and the logic of certain functions are provided.

3.1 Architecture between BERT and ViLT

3.1.1 A Deep Dive into PictoBERT

BERT BERT, as described in Devlin et al.'s work (2019) [9], is designed to condition both left and right contexts simultaneously in all layers, aiming to pre-train deep bidirectional representations from unlabeled text. Consequently, the pre-trained BERT model can be enhanced with just one additional output layer to provide state-of-the-art models for various tasks, such as question-answering and language inference, without necessitating significant task-specific architectural changes.

The model is pre-trained using unlabeled data across a range of pre-training tasks. The pre-trained parameters form the basis of the model, and labeled data from downstream tasks is used to fine-tune each parameter.

BERT is a multi-layer bidirectional Transformer encoder available in two model sizes: base (L=12, H=768, A=12, Total Parameters=110M), which shares the same size as the OpenAI GPT model, and large (L=24, H=1024, A=16, Total Parameters=340M). It has long been understood that increasing the model size leads to continuous improvements in large-scale tasks such as machine translation and language modeling.

BERT transformer employs bidirectional self-attention, whereas the GPT Transformer uses constrained self-attention, allowing each token to attend only to the context on its left. A "sequence" here refers to the input token sequence, which can be a single sentence or two sentences combined. It utilizes WordPiece embeddings (Wu et al., 2016 [53]) with a vocabulary of 30,000 tokens. Each sequence begins with a special classification token [CLS]. The final hidden state corresponding to this token serves as the aggregate sequence

representation for classification tasks. The input representation for each token is constructed by summing the respective token, segment, and position embeddings, see Figure 3.1.

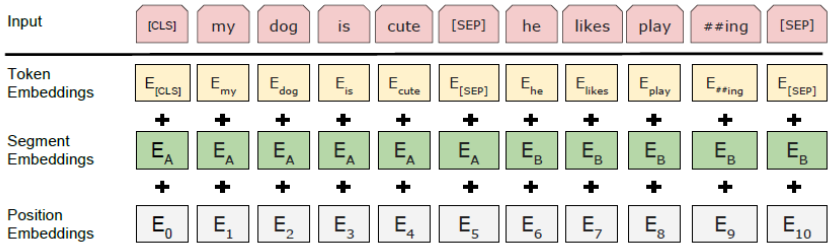


Figure 3.1: Workflow chart exploiting BERT input embeddings are the sum of the token embeddings, the segmentation embeddings, and the position embeddings.

BERT is pre-trained through two unsupervised tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). A deep bidirectional model has proven to be more powerful than a left-to-right model. The MLM procedure is employed to train a deep bidirectional representation, wherein a portion of input tokens is randomly masked, and predictions are made for these masked tokens. The final hidden vectors for the masked tokens are passed through an output softmax over the vocabulary, similar to a standard LM.

In all experiments, approximately 15% of all WordPiece tokens in each sequence are randomly masked. It's important to note that not all "masked" words are replaced with the [MASK] token. The training data generator randomly selects 15% of token positions for prediction. If the i-th token is chosen, it's replaced with the [MASK] token 80% of the time, a random token 10% of the time, or remains unchanged 10% of the time.

To train a model that comprehends sentence relationships, the model is pre-trained for a binarized next-sentence prediction task. Specifically, when selecting sentences A and B for each pretraining example, 50% of the time B

is the actual next sentence following A, while 50% of the time it's a random sentence from the corpus. The pretraining dataset comprises BooksCorpus [1] and English Wikipedia. [2]

Fine-tuning is straightforward due to the self-attention mechanism in the Transformer, which enables BERT to model various downstream tasks, whether they involve single text or text pairs, by replacing the appropriate inputs and outputs.

BERT uses the self-attention mechanism to unify these two stages because encoding a concatenated text pair with self-attention effectively incorporates bidirectional cross-attention between the two sentences. For each task, task-specific inputs and outputs are integrated into the model, and all parameters are fine-tuned end-to-end. The solutions make use of the `BertForMaskedLM` model. [3]

PictoBERT: The Evolution The PictoBERT contextualized and gloss model, after updates, comprises 317 million trainable parameters and uses token sequences that pass through an embedding layer, encoding tokens into vector representations through an attention mechanism, resulting in 1024-dimensional vector representations for each input token. The base pre-trained LM for PictoBERT is bert-large-uncased, [4] with the following parameters (see Figure 3.2):

- 24-layer;
- 1024 hidden dimension;
- 16 attention heads;
- 336M parameters.

[1] BooksCorpus
[2] English Wikipedia
[3] `BertForMaskedLM` on Hugging Face
[4] BERT large uncased on Hugging Face

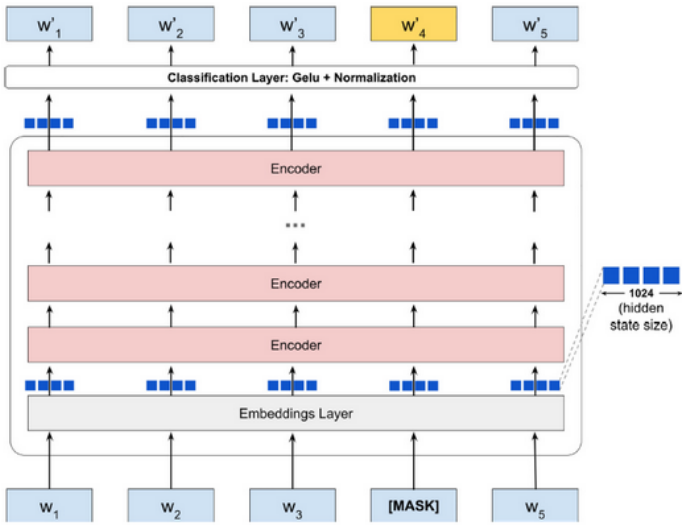


Figure 3.2: BERT Encoder architecture: inputs are encoded as vectors via the embedding layer, which subsequently sends those vectors to the encoder layers. In BERT-large, the size of the hidden state is 1024, instead of 768 in BERT-base.

For classification tasks, the model utilizes vector representations to approximate vocabulary tokens, generating a probability distribution across the vocabulary. The output includes a sequence of vectors aligned with input tokens, including the vector associated with the ”[MASK]” token. During training, BERT’s pre-trained weights are loaded, and the training loss function for BERT, calculated over 15% of randomly selected tokens, is cross-entropy.

3.1.2 Embracing Multimodality: Analysis of PictoViLT

In the realm of language and vision, the pre-train-and-fine-tune paradigm has evolved, giving rise to a category of models known as VL Pre-training (VLP). VLP models undergo pre-training involving Image Text Matching (ITM) and MLM objectives on images paired with their corresponding textual descriptions. Following this pre-training phase, they are fine-tuned for VL tasks that necessitate the integration of two modalities.

ViLT represents a significant departure from traditional Transformer architectures like BERT. Notably, its handling of visual data is devoid of convolutional layers, making it as straightforward as processing textual inputs. This characteristic confers upon ViLT a remarkable speed advantage, making it several times faster than its counterparts in the VLP domain.

The initial step in ViLT's processing involves segmenting the input image into patches and flattening it. Subsequently, the embeddings for textual and visual modalities are combined through vector addition with their respective mode-specific embedding vectors. This merged sequence is then passed through linear projection and positional embedding.

The vector representing the context, denoted as z undergoes iterative updates across D-depth Transformer layers until the final contextualized sequence is achieved. Additionally, the pooled representation of the image, which encapsulates the entire multimodal input, is obtained through linear projection and hyperbolic tangent applied to the first index of the sequence.

Within the Visual Embedding Schema, both textual and image data undergo linear embedding. The input image is sliced into patches and flattened as shown in the architecture overview of Figure 3.3.

Images are transformed into patches, word token sequences are encoded into vector representations using an attention mechanism, and text and image embeddings are unified into a sequence referred to as $z - zero$. Each input token within this sequence is represented as a 768-dimensional vector.

An important aspect of ViLT's architecture is the application of Layer Normalization (LN). Unlike BERT, where LN occurs after Multi-Head Self-Attention and Multi-Layer Perceptron layers, ViLT employs LN before these

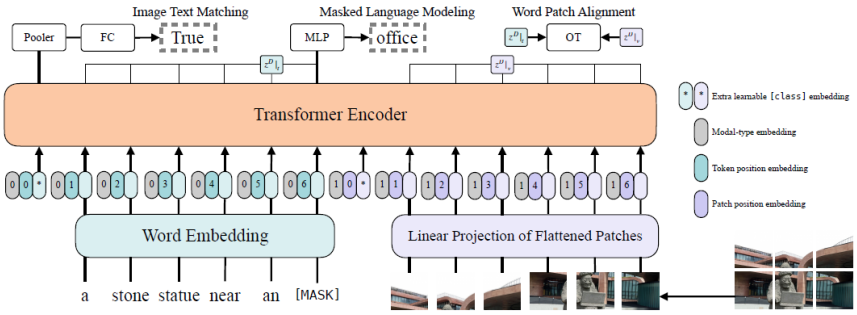


Figure 3.3: In the ViLT [21] architecture overview, the visual processing involves dividing an image into patches of fixed size, linearly embedding each patch, adding position embeddings, and subsequently feeding this sequence of vectors into a conventional Transformer encoder combined with text embeddings.

layers. This variation plays a critical role in ViLT's functioning.

ViLT-B/32 is a specific instantiation of the model with distinctive parameters, such as a hidden size (H) of 768, a layer depth (D) of 12, a patch size (P) of 32, an MLP size of 3,072, and 12 attention heads. Its training involves tasks such as ITM and MLM. ITM introduces a randomization element by replacing aligned images with different images with a 50% probability. Simultaneously, MLM tasks involve predicting the correct labels of masked text tokens based on their contextualized vector representations.

To implement this solution, ViltForMaskedLM, [5] derived from the pretrained `ViLT-B/32` model, is employed for MLM tasks. The model undergoes training for 200,000 steps, utilizing diverse datasets, including Google Conceptual Captions [44], SBU Captions [37], Microsoft COCO [29], and Visual Genome [22]. The optimization process leverages the AdamW optimizer with a base learning rate of 10^{-4} and weight decay of 10^{-2}. Learning rate scheduling incorporates a warm-up phase for the initial 10% of training steps, followed by a linear decay to zero for the remainder of training. It's important to note that customizing hyperparameters for specific tasks may yield

[5] ViltForMaskedLM on HugginFace

further improvements in downstream performance.

Lastly, as part of preprocessing, input images are resized to ensure that the shorter edge measures 384 pixels while preserving the aspect ratio by keeping the longer edge under 640 pixels.

To effectively process textual inputs, `ViLT-B/32` relies on the bert-base-uncased tokenizer for tokenization. This tokenizer ensures that text inputs are appropriately segmented into tokens, facilitating subsequent processing steps.

ViLT-B/32's patch projection results in 240 patches for images with a resolution of 384x640 pixels. During pre-training, a maximum of 200 patches are sampled. Positional embeddings (V_{pos}) withinViT-B/32 are interpolated to match individual image sizes, with patches appropriately padded for batch training. It's noteworthy that the resulting image resolution is four times smaller than the 800×1333 size utilized by other VLP models to feed their visual embedders. The textual embedding-related parameters in ViLT are initialized from scratch rather than being derived from a pre-trained BERT model. This choice is made because using pre-trained BERT parameters solely for text embeddings led to weaker performance in VL tasks [21]. ViLT-B/32 is pre-trained with text embeddings initialized from scratch for 100,000 or 200,000 steps using 64 NVIDIA V100 GPUs with a batch size of 4,096. For downstream tasks, training involves ten epochs with a batch size of 256 for VQAv2 and retrieval tasks, and 128 for NLVR2.

3.2 Dataset

3.2.1 SemChildes

Using word senses, as introduced by Schwab et al. (2020) [42], provides a more precise representation of concepts since single words can often be ambiguous. Therefore, we associate pictograms with word-senses, making the task akin to word-sense language modeling. Consequently, for PictoBERT it's needed a dataset that labels word-senses for nouns, verbs, adjectives, and adverbs.

To address this, the CHILDES, [6] MacWhinney [30] is turned into a multilingual corpus containing around 2 million sentences transcribed from children's speech. Given its conversational nature, it's decided to use this corpus for training. To make this feasible, a portion of CHILDES is labeled with word-senses using SupWSD (Papandrea et al., 2017 [38]). Specifically, sentences in North American (NA-Eng) are selected, resulting in a dataset called SemCHILDES, comprising 955,000 labeled sentences. The labeling involved assigning SemCHILDES summary a sense-key to each content word (i.e., verbs, nouns, adjectives, and adverbs), while functional words were retained in their original form. For example, the sentence "I am a little butterfly". was transformed into "i be%2:42:03:: a little%3:00:01:: butterfly%1:05:00::."

However, the sentences tend to be shorter, with 201,062 of them having four tokens and an average sentence length of six. This characteristic aligns with CHILDES, which features children's speech and typically yields shorter sentences compared to corpora sourced from newspapers, books, or Wikipedia articles, as used in training BERT. Additionally, SemCHILDES includes lemma and POS tag annotations for each word. It comes in two versions: NA-Eng and UK-Eng, containing sentences from British English

[6] CHILDES Wikipedia

Num	Sentence
1	troop%1:14:03:: be%2:42:03:: call%1:10:01:: .
2	watch%1:06:00:: this .
3	wow%1:10:00:: , look_at%2:31:00:: .
4	a tractor%1:06:00:: .
5	a tractor%1:06:00:: , .
6	what be%2:42:06:: this ?
7	what be%2:42:06:: this ?
8	what do%2:41:01:: you think%2:31:01:: it be%2:42:06:: ?
9	a ball%1:06:01:: !
10	what do%2:41:01:: you think%2:31:01:: it be%2:42:06:: ?
11	a ball%1:06:01:: .
12	maybe%4:02:00:: it be%2:42:06:: a ping-pong%1:04:00:: ball%1:06:01:: .

Table 3.1: Group of SemCHILDES sentences.

CHILDES.

Regarding the division of the North American part of SemCHILDES, it's split in 98-1-1 for training, validation, and testing purposes. The training and validation splits were utilized during pre-training, and the training split helped adjust the model weights, while the validation split served to assess performance during training. As in Figure 3.4, it's composed by:

- large corpus sentences;
- small sentences of mean length of 6 tokens per sentence;
- conversational form to adapt to the conversational task of predicting pictogram.

SemCHILDES summary.

Words		Sentences				
Total	Unique	Total	Max length	Min length	Mean length	Most frequent length
5,556,703	13,584	955,489	90	3	6	4, with 201,055 sentences

Figure 3.4: SemCHILDES summary.

Used to train Word Level Tokenizer which splits tokens in a sentence by whitespace. Sentences of 4 tokens are the most frequent, appearing 201,062 times. Each phrase is made up of word senses that are interspersed with pronouns and punctuation as in Table 3.1. For each sentence has been counted the number of tokens inside it on the right column of Figure 3.4 and is reported in the Table below 3.2.

Num tokens	Num Sentence
4	201,062
5	178,838
3	164,641
6	131,612
7	92,104
...	
64	1
74	1
59	1
86	1
65	1

Table 3.2: SemCHILDES dataset: the number of tokens per sentence is shown on the left column, and the number of sentences with that amount of tokens is displayed on the right column.

The bar graph in Figure 3.5 highlights a concentration of sentences in the first 10 x-axis locations by relating the number of sentences on the y-axis to the number of tokens per phrase on the x-axis.

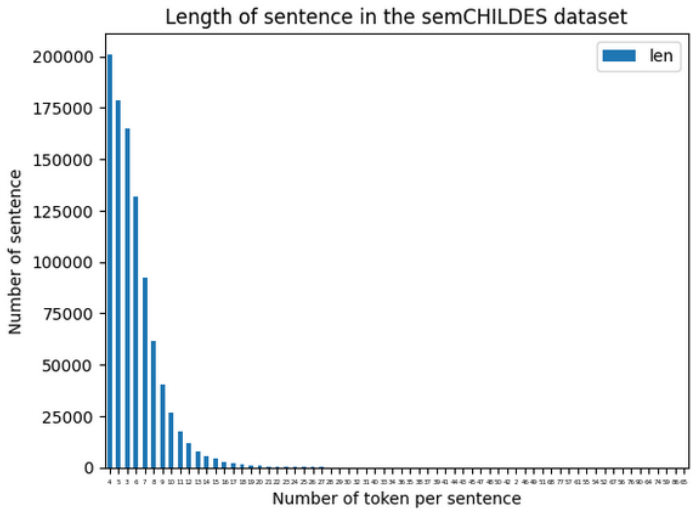


Figure 3.5: Graph distribution of sentence length in SemCHILDES dataset.

Table 3.3 shows the dataframe dataset with examples of the first 6 sentences and their number of tokens.

To map ARASAAC pictograms to WordNET 3.0 word-senses, there is a CSV file with the association between word, word-sense, and pictogram id as in Figure 3.4. SemCHILDES is adapted to fit the ARASAAC vocabulary of

Sentence	len
troop%1:14:03:: be%2:42:03:: call%1:10:01:: .	4
watch%1:06:00:: this .	3
wow%1:10:00:: , look_at%2:31:00:: .	4
a tractor%1:06:00:: .	3
a tractor%1:06:00:: , .	4
what be%2:42:06:: this ?	4

Table 3.3: SemCHILDES table of sentence and their length. It's a dataframe with 955,489 rows x 2 columns.

Unnamed: 0	Unnamed: 0.1	word	pictogram_id	synset	word_senses	Unnamed: 6
0	0	pavement	2247.0	04215402-n	pavement%1:06:01::	NaN
1	1	sidewalk	2247.0	04215402-n	sidewalk%1:06:00::	NaN
4	4	carpet	2249.0	04118021-n	carpet%1:06:00::	NaN
5	5	rug	2249.0	04118021-n	rug%1:06:00::	NaN
6	6	pillow	2250.0	03938244-n	pillow%1:06:00::	NaN

Table 3.4: Word-sense - ARASAAC mapping table.

length 8,259, deleting tokens not in the "word-sense" column vocabulary of length 3,880.

3.3 Other Datasets

The SemChildes dataset was used to transition from word-senses to words through the ARASAAC mapping for correlation definition. However, several challenges and issues arose during this process:

- Not all word-senses present in SemChildes are covered by the ARASAAC mapping and were therefore discarded;
- In cases where a word-sense had a corresponding word and a pictogram ID in the ARASAAC mapping, it did not always have a corresponding pictogram ID in the pictogram directory. Consequently, there was not a pictogram available for every word;
- Some words corresponded to multiple word-senses, but they shared the same pictogram ID, resulting in a loss of singularity.

To ensure the quality of the data, the following data-cleaning operations were performed:

Dataset	Num. Sentences
Train	555,359
Validation	5,667
Test	5,667

Table 3.5: Training, validation and test dataset split.

Unnamed: 0	Unnamed: 0.1	word	pictogram_id	synset	word_senses	Unnamed: 6
27074	27074	daddy	31146.0	0998806-n	daddy%1:18:00::	NaN
27076	27076	daddy	31146.0	10080869-n	father%1:18:00::	NaN

Table 3.6: Discrepancy between word and word-sense.

- Removal of sentences containing `proper_noun`;
- Handling of NaN values in the word-sense column;
- Elimination of duplicated sentences.

The dataset comprises a total of 566,693 sentences, which were divided into training, testing, and validation datasets. The dataset dimensions are as in Table 3.5.

Figure 3.6: Daddy pictogram.

As illustrated in Table 3.6, certain words have the same meaning and share a pictogram ID, as seen in Figure 3.6, yet they possess different values in the word-sense column.

Other datasets considered for experimentation included the Corpus for Knowledge-Enhanced LM Pre-training (KELM) [1], [7] the Constrained Text Generation Challenge for Generative Commonsense Reasoning (CommonGen)

[7] KELM Hugging Face

dataset [28], [8] and Conceptual Captions [44]. [9]

3.3.1 Image Captioning

Image captioning datasets serve as invaluable resources in the realm of CV and NLP, bridging the visual and textual domains. These datasets are carefully curated collections of images paired with descriptive captions, and they play a pivotal role in training and evaluating models capable of understanding and generating textual descriptions for visual content.

One such noteworthy dataset is Conceptual Captions, a vast repository comprising approximately 3.3 million images meticulously annotated with captions. What sets Conceptual Captions apart from many other image captioning datasets is its unique origin. Instead of adhering to the curated style of traditional image caption annotations, the images and their associated raw descriptions are harvested directly from the vast expanse of the internet. Consequently, these captions represent a wide spectrum of styles, reflecting the diversity of content available online.

The process of constructing Conceptual Captions involves the extraction of raw descriptions from the Alt-text HTML attribute associated with web images. Subsequently, these descriptions undergo a sophisticated automatic pipeline. This pipeline is meticulously designed to extract, filter, and transform candidate image/caption pairs. The ultimate aim is to strike a delicate balance between the cleanliness, informativeness, fluency, and learnability of the resulting captions.

Conceptual Captions is not merely a repository of images with captions; it is a treasure trove of insights into the world of image captioning. In the

[8] CommonGen Hugging Face
[9] Conceptual Captions Hugging Face

sections that follow, we delve into a comprehensive analysis of this dataset, focusing primarily on the captions themselves. Our analyses provide fascinating insights, such as the distribution of caption lengths. For instance, the shortest sentence in Conceptual Captions comprises a mere three tokens, while the longest stretches to an impressive 39 tokens. On average, a caption in this dataset consists of approximately 10.4 tokens, as evidenced by the data in Figure 3.9.

This observation is further validated by the dotted red line in the bar chart depicted in Figure 3.8, which underscores the mean token length of Conceptual Captions. Furthermore, Figure 3.10 offers a glimpse into the density of sentences within various token length ranges, revealing that the majority of sentences fall within the 5-15 token range.

For a more nuanced perspective, we also examine the distribution of pictogram density per Conceptual Captions sentence in Figure 3.7. This analysis provides valuable insights into how pictograms are distributed across the dataset, shedding light on their contextual usage within the captions.

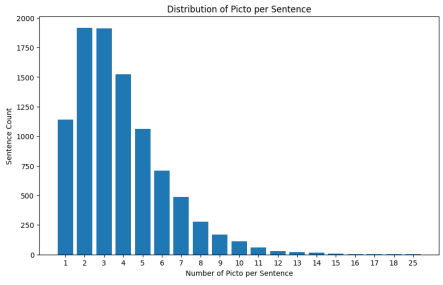


Figure 3.7: Distribution of pictogram density per Conceptual Captions sentence.

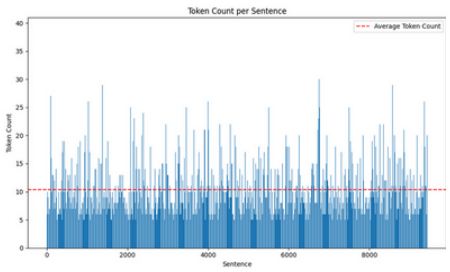


Figure 3.8: Conceptual Caption token count per sentence with average token count.

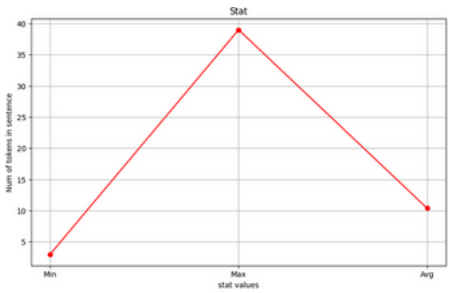


Figure 3.9: Statistical values over Conceptual Captions dataset. Min, max, and average of token count in the dataset's sentences.

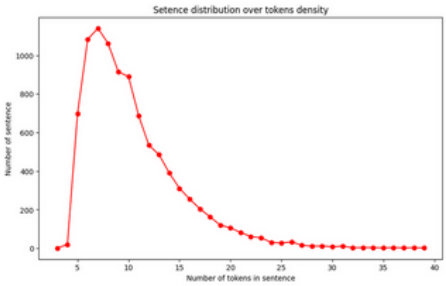


Figure 3.10: Conceptual Captions sentence distribution over tokens density.

3.3.2 Verbalized Knowledge Graphs

Verbalized Knowledge Graphs, or Verbalized KGs for brevity, represent an innovative and transformative approach in the domain of data-to-text generation. This exciting field involves the art and science of transforming structured knowledge graph triples, typically in the format of (subject, relation, object), into coherent and human-readable natural language sentences. The driving force behind Verbalized KGs is to bridge the semantic gap between raw data in structured form and the expressive power of human language.

KELM Among the remarkable datasets that harness the capabilities of Verbalized KGs, KELM dataset stands out. This dataset is a testament to the synergy between structured knowledge and natural language, comprising English KG data expertly converted into paired natural language text.

The KELM dataset is truly expansive, encompassing an impressive 18 million sentences. These sentences correspond to approximately 45 million triples derived from the knowledge graph, illustrating the richness and depth of the underlying data. Within KELM, you'll encounter an impressive repertoire of 1,500 distinct relations, underscoring the diversity and breadth of the knowledge encapsulated in this corpus.

Notably, KELM goes beyond mere data-to-text generation. The generated sentences within this dataset have undergone meticulous preprocessing, and each has been thoughtfully associated with corresponding pictogram IDs. This strategic linkage opens up exciting possibilities for multimodal applications that fuse textual and visual information.

For KELM, the generated sentences have been preprocessed and associated with pictogram IDs. The data frame contains the sentence, pictogram ID's list, and tokens of the sentence found in `pic_map["word"]`. The sentences undergo a series of preprocessing steps. These steps are designed to enhance readability, coherence, and linguistic fidelity. The preprocessing phases include:

- **Conversion to lowercase**: Standardizing the text for consistency.
- **URL removal**: Eliminating web addresses for clarity.
- **Mention removal**: Removing mentions to maintain focus.
- **Genitive replacement**: Substituting genitive ('s) with "of" for precision.
- **Parentheses removal**: Enhancing sentence flow by omitting parentheses and their contents.
- **Dash transformation**: Replacing dashes with spaces to improve legibility.
- **Abbreviation expansion**: Converting acronyms into their full forms for clarity and context.
- **Symbol removal**: Eliminating unnecessary symbols and maintaining linguistic purity.
- **Verb lemmatization**: Standardizing verbs for consistency, while retaining punctuation.

KELM sentences exhibit distinctive characteristics that offer insights into their structure and composition. On average, a sentence in the dataset consists of approximately 13.84 tokens, as depicted in Figure 3.11.

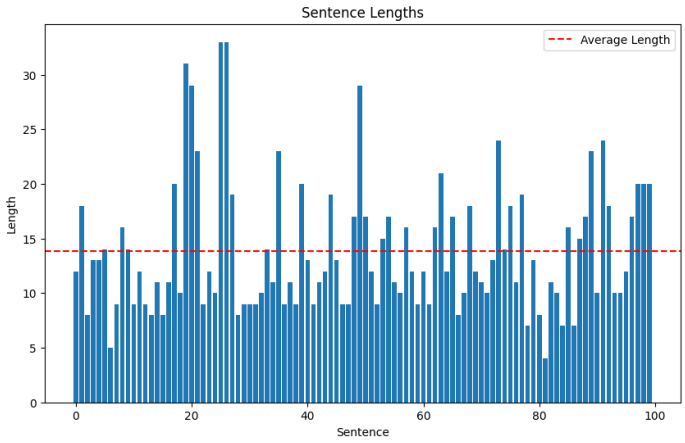


Figure 3.11: KELM sentence length.

Furthermore, the mean number of tokens per sentence hovers around 6.96, highlighting the dataset's versatility and suitability for various natural language tasks (Figure 3.12).

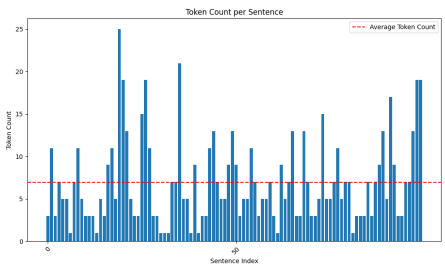


Figure 3.12: KELM token count per sentence with average token count.

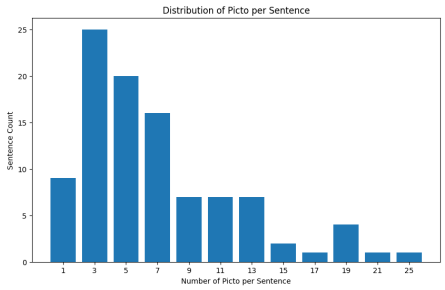


Figure 3.13: Distribution of pictogram density per KELM sentence.

For a deeper understanding of sentence structure, Figure 3.13 presents a bar chart illustrating the distribution of sentence lengths in terms of pictograms. Strikingly, a significant portion of sentences in KELM comprises

exactly three pictograms, showcasing a unique structural characteristic of this dataset.

CommonGen CommonGen is a task designed for evaluating machines' capabilities in generative commonsense reasoning, and it is associated with a benchmark dataset tailored for this purpose. The primary objective is to assess the ability of machines to generate coherent sentences describing everyday scenarios using a predefined set of common concepts.

In the context of this study, we delve into an analysis of the target column within the CommonGen dataset. Notably, this analysis focuses on the characteristics of the generated sentences

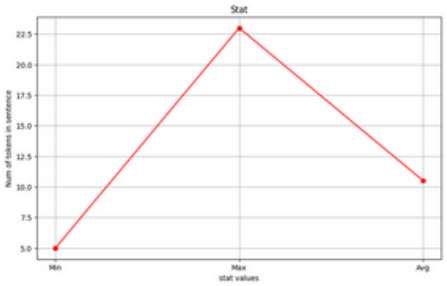


Figure 3.14: Statistical values over CommonGen dataset. Min, max, and average of token count in the dataset's sentences.

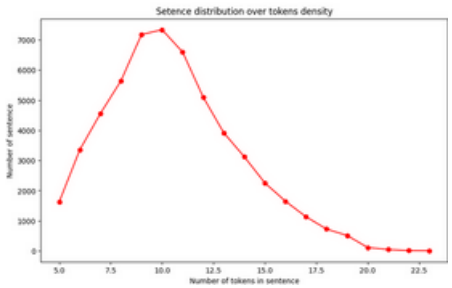


Figure 3.15: CommonGen sentence distribution over tokens density.

The dataset encompasses sentences with varying lengths. The shortest sentence in the dataset comprises 5 tokens, while the longest extends to 23 tokens. On average, sentences consist of 10.5 tokens, as illustrated in Figure 3.14.

This average sentence length is further supported by the dotted red line in the bar chart presented in Figure 3.16. Additionally, Figure 3.15 reveals that

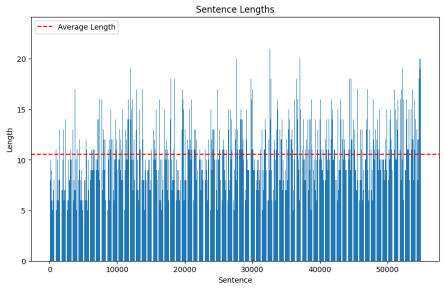


Figure 3.16: CommonGen token count per sentence with average token count.

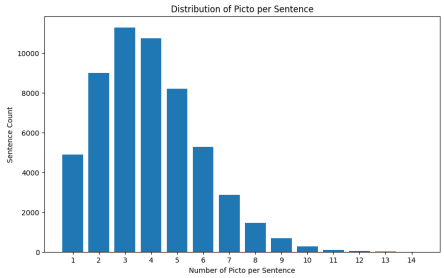


Figure 3.17: Distribution of pictogram density per CommonGen sentence.

the majority of sentences cluster in the 7.5-12.5 token range. Furthermore, Figure 3.17 illustrates the distribution of pictogram density within CommonGen sentences, providing valuable insights into the dataset's composition.

To better understand this distribution, it's essential to consider percentiles as well. For instance, 25% of the dataset contains sentences with two or fewer tokens, while the median sentence length falls at four tokens. Additionally, 75% of the data includes sentences with five tokens or fewer. At the end, we proceed with CommonGen for describing everyday scenario sentences.

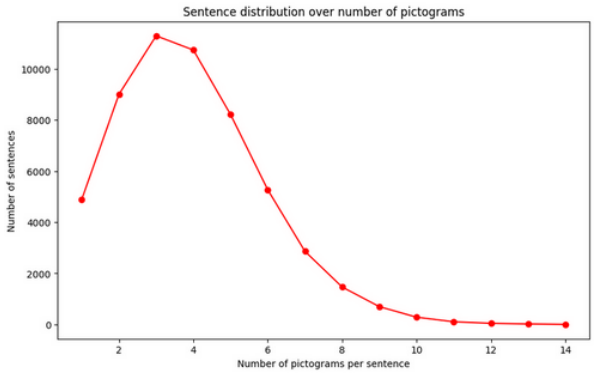


Figure 3.18: CommonGen sentence distribution over numbers of pictograms.

Analysis of images is generally represented in bar chart 3.20. Figure 3.18 shows there aren't sentences with more than 14 pictograms, here is a mean of 4 pictograms assigned to a sentence in Table 3.7.

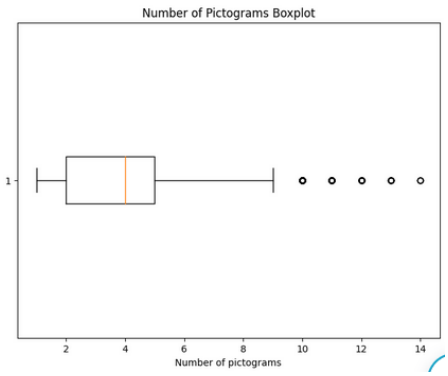


Figure 3.19: CommonGen number of pictograms boxplot.

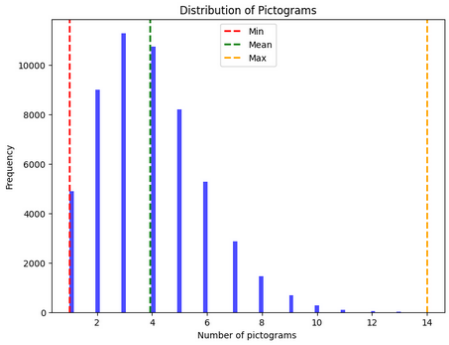


Figure 3.20: CommonGen frequency of pictograms number with Mean, Min and Max statistics.

Statistics	Values
Count	54,890.00
Mean	3.92
Std	1.94
Min	1.00
25%	2.00
50%	4.00
75%	5.00
Max	14.00

Table 3.7: Statistical Summary of Pictogram Assignments per Sentence.

The minimum is 1 and the max is 14, 25% of sentences have <=2 pictograms, 50% have <= 4 pictograms and 75% of sentence has <=5 pictograms shown in boxplot Figure 3.19.

For the fine-tuning of PictoViLT, the training dataset was downsampled to 10,000 items, the validation dataset to 2,000 items, and the test dataset to 50 items. This downsampling was done to reduce execution time, with the training set comprising approximately 83% of the total data, the validation set approximately 17%, and the test set approximately 0.4%.

3.4 Text Data Processing

Regarding text data processing for PictoViLT, a critical component is the `ViltProcessor`, combined with the pretrained `dandelinvilt-b32-mlm` model. [10] This versatile processor seamlessly combines the functionalities of `BertTokenizerFast`, [11] and `ViltImageProcessor`,[12] streamlining the text and image data preparation pipeline.

The `ViltProcessor`, [13] as depicted in the Hugging Face documentation, efficiently handles both textual and visual input, encoding and transforming them into a compatible format for downstream tasks. It operates with a vocabulary of considerable size, boasting 30,522 unique tokens, ensuring its effectiveness in processing diverse textual data.

Text encoding with `ViltProcessor` adheres to specific parameters. Padding is applied to ensure a consistent maximum length of 40 tokens, and truncation is employed to manage text length, maintaining the desired format for model input.

What makes this text-processing pipeline even more remarkable is the incorporation of a customized vocabulary. This custom vocabulary, derived from a combination of NLTK tokenizer and `ViltProcessor` tokenizer, is tailored to our dataset's unique linguistic characteristics. [14]

The custom vocabulary is constructed by including all the distinct words present in our dataset, each tokenized using the NLTK tokenizer. This vocabulary serves as the foundation for a word-ID dictionary, as illustrated in Table

[10] ViLT on HuggingFace
[11] BertTokenizerFast
[12] ViltImageProcessor
[13] `ViltProcessor` on Hugging Face
[14] NLTK documentation

3.21, facilitating seamless mapping between words in our custom vocabulary and their corresponding IDs in the `ViltProcessor`'s extensive vocabulary.

Furthermore, our customized vocabulary streamlines the data by removing words and symbols that do not find application in our specific solution. This selective pruning can be observed by comparing it with the older vocabulary, as showcased in Table 3.22.

Word	ID
determine	5646
casualties	8664
swords	10689
module	11336
skate	17260
shop	4497
dangers	16796
shelf	11142
sermon	18408
herd	14906

Figure 3.21: Customized word-ID dictionary.

Word	ID
##	30307
expeditionary	15372
unused74	750
##rite	17625
keeps	7906
whoa	23281
unused971	976
1857	8204
##ux	5602
cadiz	26342

Figure 3.22: ViltProcessor processor vocabulary.

Overall, the integration of the `ViltProcessor`, coupled with the creation of a tailored vocabulary, plays a pivotal role in optimizing text data processing for PictoViLT. These components enable efficient encoding, tokenization, and customization, laying the foundation for robust multimodal understanding and communication.

3.5 Masking

3.5.1 PictoBERT

During training During the training of PictoBERT, a robust masking strategy is implemented to enhance the model's language understanding.

The first step involves loading a trained tokenizer specifically designed for the task. This tokenizer defines special tokens like [MASK], [CLS], and [SEP] and plays a pivotal role in token manipulation during training.

The dataset, previously split and prepared, is loaded for training. It comprises sequences of tokens, attention masks, and special tokens" masks, which are fundamental for further processing.

Data collation is a critical phase where the masking strategy comes into play. [15] The `DataCollatorForLanguageModeling` is employed, [16] which replaces a fraction of tokens (controlled by MLM probability 0.15) in each batch with the [MASK] token.

Data loaders are configured to manage batch processing and apply the data collation. These loaders streamline the efficient training of PictoBERT.

During ARASAAC fine-tuning The masking strategy plays a pivotal role in PictoBERT's ability to comprehend and generate ARASAAC-enriched textual descriptions. It involves randomly masking a fraction of tokens within input sequences, prompting the model to predict these missing tokens. The choice of parameters in the masking strategy is carefully calibrated to optimize the model's performance.

Notably, the `MLM_PROBABILITY` parameter is set to 0.15, reflecting a deliberate choice to mask approximately 15% of tokens within each input sequence. This aligns harmoniously with established practices, ensuring an appropriate balance between masked and unmasked tokens to facilitate meaningful learning.

[15] Data Collator on Hugging Face
[16] `DataCollatorForLanguageModeling`

Furthermore, tokens are masked using the [MASK] token, serving as a placeholder for the tokens to be predicted. The code intricately manages the token replacement process, preserving the structure and semantics of ARASAAC-enriched sentences while enabling the model to learn the relationships between words and pictograms effectively.

Additionally, the code accounts for label generation, whereby labels corresponding to the original tokens are created and masked appropriately. This nuanced approach ensures the model's training process is well-guided, enabling it to excel in generating accurate and contextually relevant textual descriptions for multimodal inputs.

3.5.2 PictoViLT

`VILTDataset`, an integral component of our architecture, is designed as a specialized subclass of `torch.utils.data.Dataset`. This dataset plays a pivotal role in VL models, which require a harmonious fusion of textual and visual data during the training process. Here, we delve into the inner workings of this dataset, shedding light on its core functionalities.

Tokenization and Pair Generation Our `ViltDataset` class operates on sentences, treating them as the raw material for model training. The first step involves processing and tokenizing these sentences. When a match is identified between a word in the sentence and an associated image, the magic truly unfolds. In such cases, we embark on the creation of text-image pairs, introducing masked tokens that are integral to the model's learning process. However, it's important to note that phrases devoid of matches are gracefully discarded, ensuring that our training data remains relevant and purposeful.

Tokenization and Variable Initialization The crux of our dataset lies in the tokenization of sentences using the processor's tokenizer. This process entails the initialization of several vital variables, including pid (pictogram ID), target labels, masked sentences, and masked tokens. These variables are instrumental in shaping the subsequent steps of our data preparation.

Mapping and Token Matching The next phase involves a meticulous search for token indices within the sentence that corresponds to words present in a data frame. This search relies on the intricate mapping between words and pictogram IDs. If successful matches are identified:

- In cases where a solitary match is found, we pinpoint the matching index with precision;
- When multiple matches occur, a random index is thoughtfully selected from the available options. This random selection is facilitated through a method that returns a randomly chosen element from the provided sequence.

Pictogram ID Retrieval and Masking With the matching word in our grasp, we proceed to retrieve the corresponding pictogram ID. However, before proceeding further, we perform a crucial check to ensure that the image path associated with the pictogram exists. Only when this path is confirmed to exist do we initiate the masking process. The token that corresponds to the pictogram is discreetly masked, setting the stage for the model's assimilation of this masked information.

Encoding and Key Components The result of these intricate steps is the creation of an encoding object, which serves as the input to our model. This encoding object comprises several key components:

- **Input IDs**: This section contains the token IDs of the text after tokenization.

- **Attention Mask**: This component is pivotal for guiding the model's attention during processing, indicating which tokens should be considered (with a value of 1) and which should be ignored (with a value of 0).
- **Pixel Values**: Here, we encapsulate the normalized pixel values of the associated image, facilitating their integration into the model's understanding.
- **Position IDs**: This section provides crucial information about the positions of tokens within the text, enabling the model to decipher the context accurately.
- **Labels**: For specific tasks, such as text mask labeling, this segment contains token IDs that guide the model's predictions and learning.

Masking Strategies Within our dataset, we employ various masking strategies for fine-tuning, each tailored to specific learning objectives. These strategies include:

- **Logic 1** - masking 1 token with [MASK];
- **Logic 2** - masking 1 token with [MASK], another vocab word or left unchanged;
- **Logic L3** - masking multi-tokens.

Logic 1 is the simplest approach that involves the masking of only the first word associated with a pictogram ID. It forms the foundation of our masking logic.

Logic 2 is more complex. An array of tokens to mask, a list containing all the tokens in the tokenized sentence that have an associated image, will be tokens candidates for masking. Next, it's checked whether the tokens to mask list is not empty with if tokens to mask. This is a check to make sure there are

tokens available for masquerading. If it's not empty, a token from this list is randomly selected with a random choice method over the list. This token will be the token to mask. The index of the token to be masked in the tokenized sentence is then determined. The pictogram ID associated with the token to be masked is retrieved from the mapping table.

A random number is generated with the random function to determine what masking action to apply to the token. If the number is less than 0.8, the token is replaced with "[MASK]". This happens in 80% of cases. If the number is between 0.8 and 0.9, the token is replaced with a random token taken from the list of custom vocabulary keys. This happens in 10% of cases and the rest of 10% of the time, keep the token unchanged. The tokenized sentence is changed to masked tokens based on the conditions described above. The matched indices list is filtered to exclude tokens [CLS], [SEP], and [PAD] ensuring that that these special tokens are not considered for masking.

In multi-masking of **Logic 3**, the sentence is tokenized, tokens with images, and in the mapping table are put into "tokens with images". The percentage of tokens with image are chosen to be replaced in the code, and the number of tokens is rounded up to the nearest integer. Of the chosen tokens, is the logic of BERT on how to mask the tokens. The selected token is replaced with [MASK] 80% of the time, replaced with a randomly selected token from custom vocabulary 10% of the time, and left unchanged 10% of the time. The matched indices list is filtered to exclude special tokens, not useful for masking.

For each chosen token, the pictogram is stored in a list. All the pictograms in the list are concatenated and fed to the model. The percentage to mask is 80% and rounded up because some sentences have very few tokens to mask (for example 2 and with percentage=40%, the number of tokens is equal to 0.8

Logic	Sentence	Masked Tokens	Masked Sentence
1	young patient clinic	1, young	[MASK] patient clinic.
2	airplanes be taxi take airport	1, taxi -> bilingual	airplanes be bilingual take airport.
3	white domestic goat rest on grass	4, goat rest on grass	white domestic [MASK] [MASK] [MASK] [MASK]

Table 3.8: Masking examples on 3 different sentences for each model version.

and so 0 tokens are masked). This code randomly selects a subset of tokens to mask by the specified percentage, gets their indices into the tokenized phrase, replaces the tokens with ”[MASK]” or a random token based on a randomly generated value, and returns the modified tokenized phrase as a string. The 2 characteristics are a fixed percentage of masking and co-linking images along the x-axis into a single image.

Table 3.8 shows some masked sentences for the 3 logics.

3.6 Image Processing

In the domain of image processing, specifically tailored for PictoViLT, a series of carefully orchestrated steps are undertaken to prepare the original pictogram images for multimodal analysis. These actions ensure uniformity and optimal integration with the accompanying textual data.

The journey begins with the original pictogram images, each boasting dimensions of 500x500. To align them with the default picture dimension expected by the `ViltProcessor`, they are efficiently scaled down to 384x384, enabling seamless compatibility.

Subsequently, the scaled images are divided into patches, each sized at 128x128, organized in a 3x3 grid pattern. This partitioning strategy allows for a more granular analysis of the images, enhancing the model’s understanding of their visual components. Now, the encoded representation of these patches is generated, utilizing specific parameters. The patch size is configured at 16,

while the grid size is set at 24.

So each image is divided into 24x24 patches as in Figure 3.24 rendering

Figure 3.23: Pictogram with green background.

all pictograms uniforming all pictogram images also them with a not-white background like it in Figure 3.23.

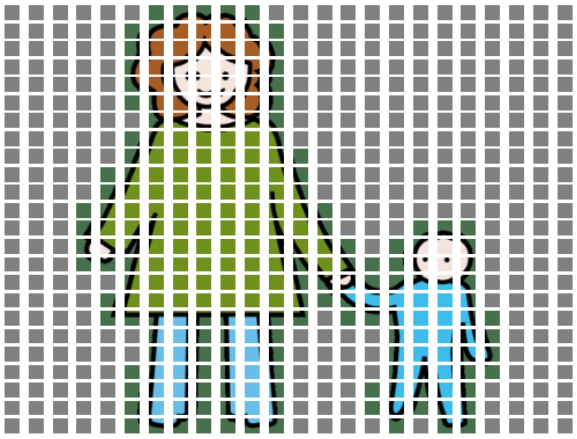


Figure 3.24: Extracted patches on pictogram.

Each patch undergoes a meticulous examination to determine its information content. Patches containing valuable information are assigned a value of 1, while those that are monochromatic receive a value of 0. Notably, patches

assigned a value of 1 retain their original color, preserving their visual characteristics. Conversely, patches with a value of 0 assume a black color, effectively serving as the background. Every pixel without useful information is black and it's like the background as in Figure 3.25.

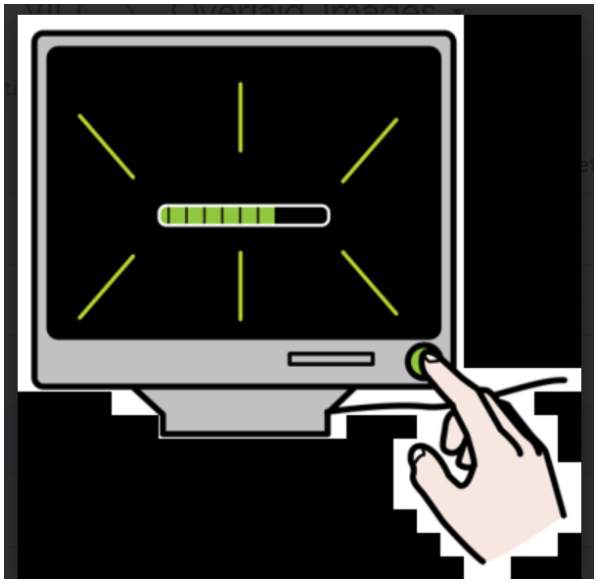


Figure 3.25: Overlaid pictogram.

The journey continues with loading these processed images from the Overlaid directory. To facilitate further analysis, the pixel values are normalized to the range [0,1] by dividing them by 255.0. Subsequently, they are transformed into torch tensors of type float32 and rearranged to adhere to the correct dimensions (C, H, W), resulting in a structured tensor representation of the image.

The true synergy of multimodal understanding emerges as text and image encoding converge. A dedicated processor adeptly manages the coexistence of textual and visual data, culminating in a holistic encoding that encapsulates both domains. Furthermore, this encoding includes a ”pixel values” field that is enriched with the image tensor, seamlessly merging textual and visual information.

In the context of masking L3, a strategic concatenation of the images follows, adhering to the order in which they appear in the tokenized sentence. This cohesive fusion is achieved after resizing the images to a uniform 384x384 dimension, ensuring harmonious integration. Specific tokens, namely [”arch”, ”on”, ”ceiling”, ”.”], are thoughtfully chosen for masking, as they are arranged in the sequence illustrated in Figure 3.26. This strategic alignment is depicted in the resulting image patches, as showcased in Figure 3.27.

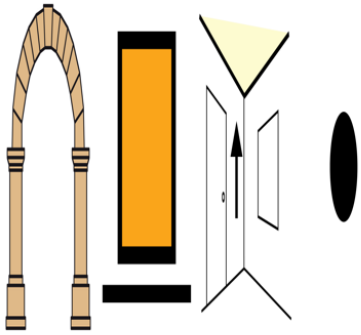


Figure 3.26: Concatenated image input for Masking Logic 3.

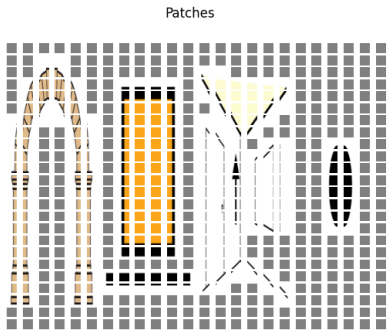


Figure 3.27: Extracted patches on the concatenated image.

This meticulous image processing pipeline for PictoViLT should harmonize visual and textual information, setting the stage for a comprehensive and multimodal understanding of the data.

3.7 Training - PictoBERT

In our training approach for PictoBERT, we adopt a set of well-considered hyperparameters, which are consistently applied to both the gloss and contextual versions of the model. These hyperparameters play a crucial role in shaping the training process, ensuring that the model can effectively learn from the data.

Firstly, we tokenize the semCHILDES dataset using a maximum token length of 32. To accomplish this, we make use of a JSON file known as `childes_all_new` as our tokenizer. We load this pretrained tokenizer with the help of the `PreTrainedTokenizerFast` [17] class from the Hugging Face Transformers library.

Training tokenizer In order to facilitate the utilization of a distinct vocabulary for BERT, the development of a new tokenizer is imperative. Tokenization, as an initial step before feeding data into a LM, involves the segmentation of sentences into individual tokens according to predefined rules, and subsequently converting these tokens into numerical representations for the model's processing. Initially, BERT employs a Word Piece tokenizer that dissects sentences into words or subwords (e.g., "playing" becomes `"play##"` and `"##ing"`).

To accommodate the integration of word-senses into the model, we embarked on training a Word Level tokenizer, which segments sentence words based on whitespace, thereby enabling the incorporation of sense keys into the process. We harnessed Hugging Face's tokenizers library for this endeavor. [18]

The construction of the tokenizer involved the following key steps:

- **Creating the Tokenizer Model**: We configured a Tokenizer using the

[17] `PreTrainedTokenizerFast` on Hugging Face
[18] Tokenizer on Hugging Face

WordLevel model, with an unknown token defined as ”[UNK]” to represent unrecognized terms.

- **Incorporating Special Tokens:** Special tokens such as ”[SEP]” (separator), ”[CLS]” (classification), ”[PAD]” (padding), ”[MASK]” (masking), and ”[UNK]” (unknown) were incorporated into the tokenizer’s vocabulary.
- **Pre-tokenization Strategy**: We employed the `WhitespaceSplit` pre-tokenizer, which splits the input text based on whitespace. [19]
- **Post-processing for Compatibility**: To ensure compatibility with BERT processing, we implemented post-processing using `BertProcessing`. [20]

  This step involved specifying token IDs for special tokens like ”[SEP]” and ”[CLS]”.

The next phase entailed training the tokenizer, which involved the following steps:

- **Word Level Training**: We utilized a `WordLevelTrainer`, with the provision of special tokens, notably ”[UNK]” to facilitate the training process.
- **Training from Examples**: The tokenizer was trained using a collection of examples, enabling it to learn the vocabulary from the provided data.
- **Vocabulary Size**: Upon completion of training, the tokenizer acquired a vocabulary size corresponding to the number of unique tokens in the training data.
- **Saving the Tokenizer**: To enable its future usage, the trained tokenizer was exported as a JSON file.

This meticulously crafted tokenizer forms a critical component of our PictoBERT model, empowering it to process text data while accommodating

[19] `WhitespaceSplit`
[20] `BertProcessing`

word-sense information for enhanced performance in word-sense language modeling tasks.

3.8 Fine-Tuning

3.8.1 PictoBERT

In this section, we delve into the intricate process of adapting a pre-trained LM, known as PictoBERT to cater to a specific vocabulary associated with ARASAAC pictograms. The overarching goal of this endeavor is to enable PictoBERT to effectively comprehend and generate text using the specialized lexicon of ARASAAC.

Data Preparation Our journey commences with the meticulous preparation of data. We import essential libraries, such as `NLTK`, requests, JSON, pandas, and more. The cornerstone of our data is the ARASAAC pictogram dataset, thoughtfully stored in a JSON file named `ARASAAC_All_pictograms`. This dataset serves as the bedrock upon which we build our adaptation process. To streamline our efforts, we initialize an empty list named `pictograms_dic` to house vital information about each pictogram, including its associated words and synsets.

Mapping ARASAAC to WordNet The heart of this adaptation lies in the intricate process of mapping ARASAAC synsets to their counterparts in WordNet—a renowned lexical database. To facilitate this, we've crafted a specialized function named `wordnet_map` that connects the dots. For each pictogram, we embark on a dual-pronged journey:

- **Personal Pronoun Category**: Pictograms classified under the "personal pronoun" category are bestowed with a collection of keywords. We graciously embrace these keywords, acknowledging their significance, and add them to our treasure trove, `pictograms_dic`.

- **Synsets Exploration**: Pictograms are also associated with synsets—a set of synonymous words or phrases. Here, we tread into the domain of WordNet. For each synset, we embark on a quest to uncover its WordNet counterpart. This connection furnishes us with a lemma—a foundational form of a word—essential for our adaptation journey. This invaluable lemma is documented in our treasure trove as well.

Saving the Mapping Once our journey through the vast realm of ARASAAC pictograms concludes, we meticulously compile `pictograms_dic` into a structured `DataFrame`. [21] This structured dataset is then bestowed with permanence in the form of a CSV file, aptly named `arasaac_mappings`.

Adaptation for SemCHILDES Our odyssey extends to SemCHILDES, another realm of linguistic exploration. To bridge the gap between ARASAAC and SemCHILDES vocabularies, we transmute the ARASAAC vocabulary to a format harmonious with SemCHILDES, ensuring linguistic continuity.

Sentence Filtering As we navigate through our linguistic voyage, we encounter textual data within a file named `all_mt_2.txt`. To ensure relevance and coherence, we meticulously filter sentences, retaining only those that align with our adapted vocabulary. Sentences bereft of relevant vocabulary or comprising solely of punctuation are gracefully excluded from our journey.

Tokenizer Creation The creation of a custom tokenizer becomes our next imperative. Leveraging the Tokenizers library, we craft a tokenizer specifically tailored to our adapted vocabulary. This tokenizer bears the hallmark of inclusion, with special tokens like ”[SEP]” and ”[CLS]” at its core.

Dataset Preparation Our filtered sentences are divided into three distinct sets: training, validation, and testing. The tailored tokenizer goes to work,

[21] DataFrame

meticulously tokenizing each sentence. The resulting tokenized data is preserved, and ready for the next leg of our journey. This transformation unfolds with precision, ensuring that each sentence is transformed into a sequence of tokens, with a maximum length of 16 tokens per sequence—a restriction that maintains efficiency and coherence.

Loading Pre-trained Models The synergy between our custom adaptation and pre-existing linguistic powerhouses takes center stage. We introduce two distinguished models: PictoBERT and BERT. The choice of the PictoBERT version, whether ”contextual” or ”gloss” offers versatility based on specific requirements. BERT a prominent pre-trained model, stands as a stalwart companion.

Loading PictoBERT Tokenizer A critical step in our journey is the incorporation of the PictoBERT tokenizer. This tokenizer, encapsulated in a JSON file, becomes an essential part of our linguistic toolkit.

Updating Embeddings Layer Our adaptation reaches a pinnacle with the harmonization of embeddings. The adapted vocabulary is meticulously aligned with the PictoBERT model. Words absent from both PictoBERT and WordNet vocabularies find solace in embeddings from an external Word2Vec source. The PictoBERT embeddings layer is updated to accommodate these adaptations, a pivotal moment in our journey.

Saving the Adapted Model Our odyssey culminates with the preservation of our meticulously adapted PictoBERT model. It receives a distinguished name, `pictobert-ARASAAC` signifying its tailored nature for the ARASAAC vocabulary. This adapted model is now poised to embark on its own linguistic adventures.

Fine-Tuning PictoBERT This process involves fine-tuning the model using carefully selected hyperparameters to ensure optimal performance in our specific linguistic domain. A recap of all the pipeline of PictoBERT's fine-tuning is in Figure 3.28.

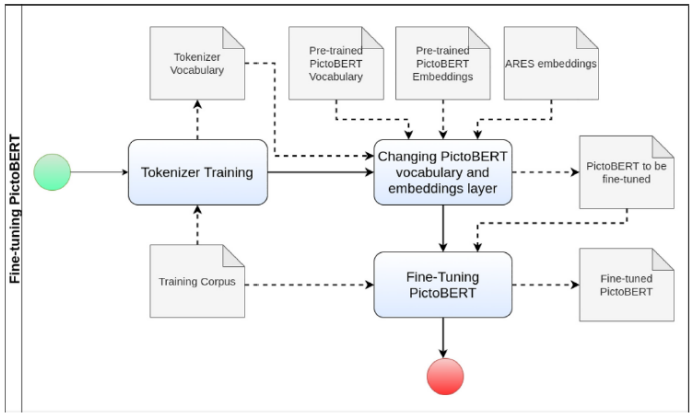


Figure 3.28: The flow diagram [40] depicts the process of fine-tuning PictoBERT for the pictogram prediction task. PictoBERT allows users to modify its language to match their own set of pictograms.

PictoBERT Adaptations for CommonGen Dataset Several key adaptations were made to ensure PictoBERT's compatibility and optimal performance with the CommonGen dataset. PictoBERT is fine-tuned on a CommonGen Wordsense dataset.

Core variables for the training process were established. These include the maximum number of training epochs, batch size, number of data-loading workers, and learning rate. Metrics such as percentiles and specific values were defined to assess the model's performance. These metrics serve as benchmarks for gauging the effectiveness of the model.

To facilitate fine-tuning, a `LitBertClassifier` class was crafted, inheriting from `pl.LightningModule`. This class encompasses a BERT-based classification model, augmented with an additional classifier tailored for the MLM

task.

Data loaders for training, validation, and testing were established. These loaders handle the loading of preprocessed data and apply appropriate collation functions, including input masking.

PictoBERT's fine-tuning commenced under the PyTorch Lightning framework. Callbacks were instituted to monitor the learning rate and save checkpoints based on validation loss, ensuring efficient training.

A comprehensive evaluation of PictoBERT's performance was conducted using a dedicated test dataset. The model's results were then compared with those of n-gram-based models for a thorough assessment. The evaluation process encompassed several stages. A test dataset was loaded from a previously saved pickle file. Various functions were defined to handle input data, including data data_collator2 and top_n_data_collator, facilitating data preparation for the model. The MyDataset class was formulated to encapsulate input data in a format compatible with PyTorch DataLoader. Two DataLoaders, namely the test_dataloader and test_dataloader_lm, were created to facilitate model testing. The PictoBERT function was employed to evaluate the performance on the test set, calculating essential metrics such as Top-N results and PPL.

A battery of tests was also conducted on the PictoBERT model using the test DataLoader. These tests included loss calculation, softmax computation to derive probabilities of masked words, and an assessment of Top-N performance. The results of PictoBERT's model performance were collected and organized into a top dictionary for further analysis.

Furthermore, information pertaining to previously trained n-gram models was loaded and stored in pickle files, enabling their utilization in evaluating

the test dataset. The evaluation encompassed various metrics, including Top-N performance and PPL, ultimately resulting in a comprehensive results list that captured the performance values of PictoBERT models and n-gram models across multiple evaluation criteria.

3.8.2 PictoViLT

Within the PictoViLT project, the journey begins with the utilization of a pre-trained model, originally designed for vision-related tasks and NLP. However, our endeavor revolves around its fine-tuning to adapt to a distinct, bespoke task, employing a carefully curated dataset tailored to our specific needs.

At the core of our adaptation process lies the model class, equipped with a suite of essential methods designed to facilitate the training and evaluation of our fine-tuned model. This class embraces key parameters, including the training data loader, the designated computing device, the learning rate, and the weight decay.

The training process unfolds as follows:

- **Model Preparation:** The training process commences with meticulous preparation. We initiate this phase by signaling the model's readiness for training using the command `self.model.train()`. This step sets the stage for the acquisition of knowledge.
- **Forward Pass and Loss Calculation**: As the training unfolds, our model engages in a meticulous examination of each batch within the training data loader. With each batch, our model undertakes a forward pass, adeptly calculating the loss associated with the input data. This loss is accumulated, contributing to the computation of the total loss.

- **Gradient Computation**: A pivotal moment arrives as a crucial backward pass is executed. During this phase, gradients are computed with precision. These gradients play a fundamental role in the refinement of the model's parameters, enabling it to learn and adapt effectively.
- **Information Documentation**: Throughout the entire training journey, we leave no detail undocumented. Critical information, such as the loss incurred at each batch and memory consumption, is meticulously recorded. This information serves as a compass, guiding our understanding of the model's performance and resource utilization.

Upon completing the training on all batches, we embark on the pivotal task of computing the average loss across the entire training `dataloader`, a metric of paramount importance.

The evaluation process, on the other hand, follows a distinct trajectory:

- **Transition to Evaluation Mode:** The evaluation process embarks on a unique trajectory. It commences with the graceful transition of the model into evaluation mode, achieved through the command `self.model.eval()`. This mode signals the model's readiness for a meticulous performance assessment.
- **Forward Pass and Calculations:** Within the confines of the validation `dataloader`, the model embarks on a purposeful journey. For each batch encountered, the model executes a single forward pass, deftly calculating both the loss and its predictions. This intricate calculation lays the foundation for rigorous evaluation.
- **Loss Integration and Accuracy Calculation:** The evaluation process is a quest for precision. As the model processes each batch, the calculated loss is meticulously integrated into the total loss, ensuring that no detail escapes scrutiny. Additionally, the model's accuracy is computed through a meticulous comparison of its predictions with the actual labels.

- **Insightful Documentation:** Similar to the training phase, the evaluation phase is not devoid of meticulous documentation. We vigilantly record valuable insights, including accuracy metrics and memory consumption. These insights serve as beacons, illuminating our understanding of the model's performance and resource utilization.

Chapter 4

Implementation

Here are reported Python code implementations of the main concepts explained in Chapter 3. Each code snippet refers to a specific section, where concepts are dealt with in a deep way.

Figure 4.1 showcases the essential hyperparameter settings employed during the training of PictoBERT. Section 3.7 lists hyperparameter sets for training, Section 3.8.1 for fine-tuning.

```
MAX_EPOCHS = 10
WARMUP_STEPS = int(MAX_EPOCHS * 0.15)
BATCH_SIZE = 112
LEARNING_RATE = 1e-04
NUM_WORKERS = 4
GPUS = torch.cuda.device_count()
PRECISION = 16 if torch.cuda.device_count() > 0 else 32
MLM_PROBABILITY= 0.15
```

Figure 4.1: PictoBERT hyperparameters settings for training.

Figure 4.2 illustrates the step of loading the PictoBERT tokenizer described in Section 3.7 for Tokenizer `childes_all_new` and for `tokenizer_arasaac` in Section 3.8.1 and Section .

Figure 4.3 provides a concise overview of the training process for PictoBERT. Section 3.7 focuses on the PictoBERT training logic shown below.

```
#the 2 tokenizer paths.
TOKENIZER_PATH = "childes_all_new.json"
TOKENIZER_PATH = "tokenizer_arasaac.json"
from transformers import PreTrainedTokenizerFast
loaded_tokenizer = PreTrainedTokenizerFast(tokenizer_file=TOKENIZER_PATH)
loaded_tokenizer.pad_token = "[PAD]"
loaded_tokenizer.sep_token = "[SEP]"
loaded_tokenizer.mask_token = "[MASK]"
loaded_tokenizer.cls_token = "[CLS]"
loaded_tokenizer.unk_token = "[UNK]"
```

Figure 4.2: PictoBERT tokenizer loading.

```
#Logger
from pytorch_lightning import loggers as pl_loggers
tb_logger = pl_loggers.TensorBoardLogger("logs",name=MODEL_VERSION)

#Checkpoint
checkpoint_callback = ModelCheckpoint(
    dirpath="checkpoints",
    filename='bert-large-{epoch:02d}-{train_loss:.2f}-{val_loss:.2f}',
    mode='min',
    monitor="val_loss",
    save_last=True)


#Trainer
trainer = pl.Trainer(
    accelerator='gpu',
    max_epochs=MAX_EPOCHS,
    logger=tb_logger,
    callbacks=[checkpoint_callback],
    precision=PRECISION)

#2 PictoBERT version models
MODEL_VERSION = "contextual"
MODEL_VERSION = "gloss"


#Call model version
model = LitBertClassifier(MODEL_VERSION)

#Train
trainer.fit(model, train_dataloader, val_dataloader)
```

Figure 4.3: Training PictoBERT code.

Figure 4.4 illustrates the management of datasets for training, validation, and testing within the PictoBERT framework. In Section 4, there are deeper references to this code.

Figure 4.6 provides the code to create data loaders used in the PictoBERT

```
class MyDataset(Dataset):
  def __init__(self, examples):
    self.input_ids = examples['input_ids']
    self.attention_mask = examples['attention_mask']
    self.special_tokens_mask = examples['special_tokens_mask']

  def __len__(self):
    return len(self.input_ids)

  def __getitem__(self, idx):
    input_ids = tensor(self.input_ids[idx])
    attention_mask = tensor(self.attention_mask[idx])

    special_tokens_mask = tensor(self.special_tokens_mask[idx])
    return {
      "input_ids":input_ids,
      "attention_mask":attention_mask,
      "special_tokens_mask":special_tokens_mask}

import pickle
tds = pickle.load(open('train_data.pt','rb'))
vds = pickle.load(open('val_data.pt','rb'))
tsds = pickle.load(open('test_data.pt','rb'))

train_dataset = MyDataset(tds)

val_dataset = MyDataset(vds)
test_dataset = MyDataset(tsds)
```

Figure 4.4: PictoBERT dataset managing.

framework, described in Section 4 and in the following Section 3.8.1.

Figure 4.6 presents a code snippet for a tokenize function used in the PictoBERT framework. It is exploited in Section and Section .

The Python function depicted in Figure 4.7 evaluates the performance of a PictoBERT model on the test data. It is referred to computed results in the following Section 5.2. It calculates the Top-N accuracy by comparing predicted and actual labels, recording losses, and sorting predictions. The function is essential for assessing the model's effectiveness and ability to provide accurate predictions.

```
from torch.utils.data import DataLoader
from torch.utils.data.distributed import DistributedSampler
from transformers import DataCollatorForLanguageModeling

data_collator = DataCollatorForLanguageModeling(
tokenizer=loaded_tokenizer, mlm_probability=MLM_PROBABILITY)
train_dataloader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    pin_memory=True,
    collate_fn=data_collator,
    drop_last = True,
    shuffle=True)

val_dataloader = DataLoader(
    val_dataset,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    pin_memory=True,
    collate_fn=data_collator,
    drop_last = True)

test_dataloader = DataLoader(
    test_dataset,batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    collate_fn=data_collator,
    pin_memory=True,
    drop_last = True)
```

Figure 4.5: PictoBERT dataloader.

```
max_len = 32
#function to tokenizer
def tokenize_function(tokenizer,examples):
    # Remove empty lines
    examples = [line for line in examples if len(line) > 0
    and not line.isspace()]
    bert = tokenizer(
        examples,
        padding="max_length",
        max_length=max_len,
        return_special_tokens_mask=True,
        truncation=True)
    ngram = tokenizer(examples,add_special_tokens=False).input_ids
    return bert,ngram
```

Figure 4.6: PictoBERT tokenize function.

```
def run_pictobert(model):
  topn = [1,^^I9,^^I18,^^I25,^^I36]
  ...
  with torch.no_grad():
    for batch in tqdm(test_dataloader):
      if torch.cuda.is_available():
        output = pictobert(batch['input_ids'].to('cuda:0'),
        ↪  labels=batch['labels'].to('cuda:0'),
        attention_mask=batch['attention_mask'].to('cuda:0'))
      else:
        output = pictobert(batch['input_ids'],
        ↪  labels=batch['labels'],attention_mask=batch['attention_mask'])

      losses.append(float(output[0].detach().cpu()))
      predictions = F.softmax(output[1], dim=-1)
      labels = batch['labels']
      masked = batch['input_ids']
      n = masked.detach().cpu().numpy()
      predicted = predictions.detach().cpu().numpy()[n ==
      ↪  loaded_tokenizer.mask_token_id]

      ids = np.argsort(-1*predicted,axis=1)

      for ti, first in enumerate(topn):
        count = 0

        for i, data in enumerate(ids):
          if labels[masked == loaded_tokenizer.mask_token_id][i].item() in
          ↪  data[:first]:
            count += 1
        isin = count/predicted.shape[0]
        topn_values[ti].append(isin)

    ...
```

Figure 4.7: PictoBERT Top-N accuracy and PPL evaluator method.

`MyModel` class in Figure 4.8 serves as a foundational blueprint for fine-tuning VL models. It encapsulates the entire lifecycle of model adaptation, encompassing training, evaluation, and the monitoring of crucial metrics such as loss and accuracy. Once the model is finely attuned, it stands ready for future applications, poised to excel in its specialized domain.

```python
class MyModel:
    ...
    def train(self, train_dataloader, device, learning_rate, weight_decay):
        self.model.train()
        ...
        optimizer = torch.optim.AdamW(self.model.parameters(), lr=learning_rate,
        ↪  weight_decay=weight_decay)
        for batch in tqdm(train_dataloader):
          batch = {k:v for k,v in batch.items()}
          optimizer.zero_grad()
          outputs = self.model(**batch)
          loss = outputs.loss
          train_loss += loss.item()
          loss.backward()
          optimizer.step()
          ...
        train_loss /= len(train_dataloader)
        #self.model.save_pretrained(save_path)
        return train_loss

    def eval(self, val_dataloader, device):
        self.model.eval()
        ...
        criterion = nn.CrossEntropyLoss()

        for batch in tqdm(val_dataloader):
          batch = {k:v for k,v in batch.items()}
          input_ids = batch['input_ids']
          attention_mask = batch['attention_mask']
          labels = batch['labels']
          with torch.no_grad():
            outputs = self.model(**batch)

          loss = criterion(outputs.logits.view(-1,
          ↪  len(processor.tokenizer.vocab)), labels.view(-1))
          val_loss += loss.item()
         ...
          max_idx = torch.argmax(outputs.logits, dim=-1)
          predicted_labels = max_idx.squeeze(-1)
          correct_predictions += (predicted_labels == labels).sum().item()
          true_labels.extend(labels.tolist())
          predictions.extend(predicted_labels.tolist())
          ...
        return validation_loss, accuracy
```

Figure 4.8: PictoViLT Model class of Logic 1 with focus on train and validation methods.

The code below, in Figure 4.9, is a representation in the Python programming language of the `VILTDataset` class used for the management and encoding of multimodal data, see Section 3.4 and Section 3.6 for fine-grained descriptions. This is the `VILTDataset` backbone for Logic 1; the other two classes are for managing masking strategies 2 and 3.

```
#class dataset of Logic 1 but there are for logic 2 and 3
class VILTDataset(torch.utils.data.Dataset):
    def __init__(self, sentence, processor, vocab, df, mapping, masked_prob=0.15,
    ↪  max_length=40):
    ...
    def __getitem__(self, idx):
        ...
        masked_sentence = tokenizer.convert_tokens_to_string(tokenized_sentence)
        matched_indices = [i for i in range(len(tokenized_sentence))
        if (pic_map["word"] == tokenized_sentence[i]).any()]
        if matched_indices:
            if len(matched_indices) == 1:
                i = matched_indices[0]
            else:
                i = random.choice(matched_indices)

            matched_rows = pic_map[pic_map['word'] == tokenized_sentence[i]]
            row = matched_rows.sample(n=1)
            pid = int(row["pictogram_id"].item())
            ...
            if os.path.exists(image_path):
                masked_token = tokenized_sentence[i]
                tokenized_sentence[i] = "[MASK]"
                masked_sentence
                ↪  =tokenizer.convert_tokens_to_string(tokenized_sentence)

                image = Image.open("Overlaid_Images/" + str(pid) + ".png")
                normalized_image = np.array(image, dtype=np.float32) / 255.0
                masked_image_float_tensor =
                ↪  torch.tensor(np.array(normalized_image),
                ↪  dtype=torch.float32).permute(2, 0, 1)
                pixel_values= masked_image_float_tensor

                #encoding
                encoding = self.processor(image, masked_sentence,
                ↪  padding="max_length", max_length=40, truncation=True,
                ↪  return_tensors="pt")
                for k, v in encoding.items():
                    if is instance(v, torch.Tensor):
                        encoding[k] = v.squeeze()
                encoding["labels"] =
                ↪  torch.tensor(self.processor.tokenizer(sentence,
                ↪  padding='max_length', max_length=40)["input_ids"])
                encoding['pixel_values']=pixel_values
                encoding['pixel_values'].squeeze(0)
                return encoding
        return None
```

Figure 4.9: PictoViLT `VILTDataset` class of Logic 1.

Here, in Figure 4.10, there is the code snippet for the instantiation of datasets for training, validation set, and test set in `VILTDataset` class and DataLoader. For details, see Section 3.5.2 and Section 3.8.2.

```
#Call of VILTDataset class. It's repeated for validation and test sets.
train_dataset =VILTDataset(
sentence=word_sentence_train,processor=processor,
vocab=vocab_custom, df=df_train, mapping=pic_map)
#dict_keys(['input_ids', 'token_type_ids', 'attention_mask', 'pixel_values',
↪  'labels'])
BATCH_SIZE = 32 #or another dimension
#during each epoch, data will be presented in a random order
#to the model with #shuffle=T
#It's repeated for validation and test sets.
train_dataloader= DataLoader(
train_dataset,collate_fn=collate_fn, batch_size=BATCH_SIZE, shuffle=True)
```

Figure 4.10: PictoViLT ViltDataset class of Logic 1 method call.

As described in Section 3.4, the logic for constructing Customized vocabulary involves NLTK Python library.

```
import nltk
nltk.download('punkt')

from nltk.tokenize import word_tokenize
# Tokenize sentences using the NLTK tokenizer
tokenized_sentences = [word_tokenize(sentence) for sentence in sw]

# Create custom vocabulary using tokenized words
vocab = set()
for tokens in tokenized_sentences:
    vocab.update(tokens)

#Creating the word-ID dictionary using the processor vocabulary
vocab_id_mapping = {}
for word in vocab:
    if word in processor.tokenizer.vocab:
        word_id = processor.tokenizer.vocab[word]
        vocab_id_mapping[word] = word_id
```

Figure 4.11: PictoViLT custom vocabulary creation.

As introduced in Section 5.1, here is the logic used to compute PPL and Top-N accuracy for PictoViLT evaluation.

The Python code snippet depicted in Figure 4.13 illustrates the inference

```python
def topn_accuracy(model, test_dataloader, topn_values, model_version):
    ...
    with torch.no_grad():
        for batch in test_dataloader:
            input_ids = batch['input_ids']
            attention_mask = batch['attention_mask']
            labels = batch['labels']
            pixel_values = batch['pixel_values']

            # Get model predictions for text
            outputs = model.model(input_ids=input_ids,
            attention_mask=attention_mask, pixel_values=pixel_values,
            labels=labels)
            total_loss += outputs.loss.sum().item()
            total_examples += input_ids.size(0)
            predictions = F.softmax(outputs.logits, dim=-1)
            masked_positions = input_ids == processor.tokenizer.mask_token_id

            #Calculate Top-N predictions
            for i, topn in enumerate(topn_values):
                topn_predictions = torch.topk(outputs.logits, k=topn,
                ↪  dim=-1).indices[:,:, :topn]
                for j, data in enumerate(topn_predictions):
                    mask_positions = torch.nonzero(masked_positions[j]).squeeze(1)
                    label = labels[j][mask_positions].unsqueeze(1)
                    data_masked = data[mask_positions]
                    if (label == data_masked).any().item():
                        topn_correct[i] += 1

    #Calculate accuracy percentages
    topn_accuracies = [c / total_examples for c in topn_correct]
    #PPL
    ppl = torch.exp(torch.tensor(total_loss / total_examples)).item()
    ...
    df = pd.DataFrame([res_dict], columns=['top-1', 'top-9', 'top-18',
    'top-25', 'top-36', 'PPL', 'Model version'])
    return df
```

Figure 4.12: PictoViLT Top-N accuracy and PPL computation method.

method within the PictoViLT model. This method is designed to generate inferences by predicting tokens and their associated scores for a given input sentence and image features. The process involves sampling the top-k tokens from the model's output, decoding them, and ranking them based on their scores. The unique tokens are tracked to ensure diversity, and the scores are both non-normalized and normalized to provide insight into token importance.

Finally, the predicted tokens, their scores, and normalized scores are printed for analysis and further use. This code segment is instrumental in understanding the inner workings of PictoViLT's inference generation process.

In Section 5.3, the inferences examples.

```python
def vilt_generate_inference(model, sentence, image_features, k):
    ..
    unique_tokens = set()  # Set to keep track of unique tokens
    for i, token in enumerate(top_k.indices):
        input_ids[0, masked_index] = token
        outputs = model.model(input_ids, pixel_values=image_features)
        logits = outputs.logits[0, masked_index, :]
        probs = F.softmax(logits, dim=-1)
        decoded_token = tokenizer.decode(torch.argmax(probs))
        if decoded_token not in unique_tokens and not
        ↪  decoded_token.startswith("["):
            predicted_tokens.append(decoded_token)
             # Add the score to the list
            predicted_scores.append(probs[token].item())
            unique_tokens.add(decoded_token)
        # Check if we have enough predicted tokens
        if len(predicted_tokens) == k:
            break
    # Sort predicted tokens and scores based on scores
    #in descending order (non-normalized scores)
    predicted_tokens_sorted, predicted_scores_sorted =
    ↪  zip(*sorted(zip(predicted_tokens, predicted_scores), key=lambda x: x[1],
    ↪  reverse=True))
    predicted_scores_sorted = [round(score, 7) for score in
    ↪  predicted_scores_sorted]
    # Normalize scores using the sum of scores
    score_sum = sum(predicted_scores_sorted)
    predicted_scores_normalized = [score / score_sum for score in
    ↪  predicted_scores_sorted]
    # Round the normalized scores to 4 decimal places
    predicted_scores_normalized = [round(score, 7) for score in
    ↪  predicted_scores_normalized]
    # Print predicted tokens and scores with textual representation
    print("Predicted Tokens and Scores:")
    for token, score, normalized_score in zip(predicted_tokens_sorted,
    ↪  predicted_scores_sorted, predicted_scores_normalized):
        print(f"Token: {token}, Score: {score}, normalized scores:
        ↪  {normalized_score}")
    return predicted_tokens_sorted, predicted_scores_normalized
```

Figure 4.13: PictoViLT generate inference method.

Figure 4.14 shows the interactive HTML method within the PictoViLT codebase. This method, part of the Python class, enables the dynamic display of pictograms and textual content in response to user interactions. It covers functions for displaying pictograms, generating inferences, and rendering HTML representations of pictograms and text. This code is integral to the interactive and user-friendly aspects of the PictoViLT system, enhancing its accessibility and usability. It is reported for PictoViLT but is exploited also for PictoBERT, as is remembered in the following Paragraph 5.2, while Figure 5.7 shows its functioning.

```
class Pictogram(object):
...
def show_pictograms():
    pictograms = {}
    #if are at step 0, it shows all pictograms
    #giving the possibility to select one
    if len(sentence) == 0:
        for i, row in arasaac_mapping.sample(32).iterrows():
            pictogram_id = row['pictogram_id']
            image_path = "Pittograms/{}.png".format(int(pictogram_id))
            if os.path.exists(image_path):
                pictogram = Pictogram(pictogram_id, label=row['word_senses'],
                ↪  word=row['word'], callback=do_something)
                pictograms[row['word']] = pictogram._repr_html_()
    else:
        text, picto_id = prepare_input(sentence)
        image_path = "Pictograms/{}.png".format(int(picto_id))
        if os.path.exists(image_path):
            image = Image.open(image_path)
            image = image.convert("RGB")
            encoding = processor(image, text, return_tensors="pt")
            print(sentence)
            #call vilt_generate_inference method
            predicted_tokens, scores = vilt_generate_inference(model, text,
            ↪  encoding.pixel_values, 16)
            ...
```

Figure 4.14: PictoViLT html interactive method.

Chapter 5

Results

5.1 Experimental setup

In this section, we delve into the experimental setup and evaluation metrics used to assess the performance of our models. We utilize the Weights and Biases platform to track experiment results, enabling comprehensive analysis of model training. [1]

PictoBERT pre-training setting PictoBERT is pre-trained using the Sem-CHILDES dataset with specific hyperparameters, including a batch size of 128 sequences, each containing 32 tokens. The optimizer used is the same as BERT, with a learning rate of 1×10^{-4}, L2 weight decay set at 0.01, and a linear decay of the learning rate. Training is performed on a 16 GB NVIDIA Tesla V100 GPU for 500 epochs, with each epoch taking approximately 20 minutes.

PictoBERT training setting During training, we formed batches comprising 128 sequences, each containing 4,069 tokens.
Random masking with mlm probability was applied to these batches using the `DataCollatorForLanguageModeling`. To optimize the training process, we

[1] wandb

employed the Adam Weigth Decay Optimizer, with a learning rate of 1e-4, and included beta1=0.9 and beta2=0.999 for effective weight updates. Furthermore, L2 weight decay with a factor of 0.01 was introduced, along with a linear decay strategy for the learning rate. Training processes for PictoBERT versions were executed on a powerful server infrastructure, specifically utilizing an NVIDIA GEFORCE 3090 graphics card.

For n-gram LM settings, a set of variables is initialized to facilitate the computation of cross-entropy and PPL. These models are created with varying orders, spanning from 2 to 10. Within a for loop that iterates across these orders, several essential steps occur. Firstly, the training data undergoes transformation into sequences of n-grams, each conforming to a specific order, while also incorporating left-padding using the ”[PAD]” symbol. Next, an n-gram LM, employing Maximum Likelihood Estimation, is instantiated, trained on the prepared n-gram training data, and then persistently stored as a file, denoted as `order-gram_model` where `order` signifies the model's order. Subsequently, the evaluation metrics are meticulously computed.

PictoBERT fine-tuning setting The hyperparameters chosen for fine-tuning are:

- **Maximum Epochs:** 10
- **Batch Size:** 32
- **Number of Workers:** 4
- **Learning Rate:** 1×10^{-6}
- **MLM Probability:** 0.15
- **Accumulate Grad Batches:** 4

This setting is referred to ARASAAC Fine-tuning and also to the CommonGen word-sense dataset for the comparison with PictoViLT. Optimizers and

learning rate reduction schedulers were configured for the training process. The choice of optimizer was AdamW, known for its effectiveness in fine-tuning transformer models. Several hyperparameters and configurations were meticulously tuned to attain optimal results during

PictoViLT Fine-tuning setting Fine-tuning with Google Colab and after on the server. [2] In the specific context of fine-tuning on Colab, we meticulously select the following hyperparameters to guide our model:

- **Learning Rate**: A judiciously chosen learning rate of 1×10^{-4} is set, strategically orchestrating the pace of our model's adaptation. This critical parameter guides the model's adjustment process with precision.
- **Epochs**: Our fine-tuning journey spans across two meticulously planned epochs. This deliberate choice ensures that our model benefits from an ample training period, allowing it to grasp the intricacies of our specialized task effectively.
- **Batch Size**: Each batch, thoughtfully constructed, encompasses 8 sequences. This batch size strikes an elegant balance between computational efficiency and the model's overall effectiveness. It facilitates efficient processing while preserving the model's capacity to capture nuances.

It's worth noting that the fine-tuning process is executed on Colab's formidable A100 GPU—a high-performance graphics processing unit crafted by NVIDIA. This powerhouse boasts a staggering 19 teraflops of computational power and over 40GB of high-bandwidth memory, rendering it an ideal companion for our computational tasks.

[2] Google Colab

Additionally, we have transitioned our computational tasks to a server environment, making use of the Docker tool for containerization. [3] This transition includes the fine-tuning of the PictoViLT model on the server, and the results of this fine-tuning process are summarized in Table 5.5. Our workflow now involves running code within these containers and utilizing the Slurm job scheduler for efficient resource management. [4] In this fine-tuning setup, we used a batch size of 32 and down-sampled the training dataset to 10,000 items, conducting training over 10 epochs. Similarly, the validation dataset was down-sampled to 2,000 items for evaluation throughout the 10 epochs.

L1 fine-tuning computations were executed on a TITAN XP graphics card while L2 and L3 fine-tuning were performed using an NVIDIA GEFORCE 3090 graphics card, which boasts 11GB of memory, and a batch size of 32.

Evalutaion metrics In the context of evaluating LMs such as n-grams, PictoBERT, and PictoViLT, two key metrics, PPL, and Top-N accuracy, play a crucial role.

PPL measures the LM's aptitude for predicting sequences by quantifying its degree of uncertainty when handling a dataset. A lower PPL signifies more consistent and accurate predictions. While theoretically ranging from 1 to infinity, practical PPL values generally span from low positives to higher digits, contingent on the model's complexity and text characteristics. Lower PPL values are generally preferred, with values close to 1 indicating high prediction confidence and adherence to the actual word distribution within the text. PPL serves as a crucial gauge of an LM's predictive quality and its capacity to capture the inherent text structure.

[3] Docker
[4] Slurm Web

In contrast, **Top-N** accuracy evaluates a model's proficiency in generating correct words within the first N predictions. For example, top-1 accuracy verifies if the first predicted word is correct, while top-8 accuracy checks if any of the first five predictions are accurate. Top-N accuracy metrics vary according to the specific N value chosen for evaluation, and these metrics are expressed as percentages ranging from 0 to 100. Higher Top-N accuracy values reflect superior model performance. Considering multiple Top-N accuracy metrics provides a comprehensive assessment of the model's capabilities, with top-1 accuracy focusing on the primary prediction, and top-5 accuracy determining if the correct answer is among the top five predictions.

When comparing PictoBERT with n-gram LMs, **cross-entropy** comes into play. This metric quantifies how effectively an LM predicts word probabilities relative to the actual word distribution in the text. Lower cross-entropy scores signify better alignment with the real-world distribution and are a widely adopted measure of a model's predictive prowess. However, in the context of comparing transformer-based models like PictoBERT and PictoViLT, cross-entropy takes a backseat as the primary metric. Instead, metrics such as Top-N accuracy and PPL are favored for assessing prediction accuracy and coherence.

These metrics are pivotal for LM evaluation, as they gauge predictive quality and coherence, facilitating performance assessment across various NLP tasks. Subsequent sections delve into the results achieved by PictoViLT in Section 5.3 and by PictoBERT in Section 5.2. Additionally, they offer a comparative analysis and insights into PictoViLT's interpretability in Section 5.3 and Section 5.3, respectively.

5.2 PictoBERT

Figure 5.1 and Figure 5.2 illustrates the training and validation loss trends for Contextualized PictoBERT, with the x-axis representing the training steps across epochs and the y-axis denoting the corresponding loss values.

Figure 5.1: Training Progress of Contextualized PictoBERT: The x-axis represents the training steps across multiple epochs, while the y-axis denotes the corresponding loss values, offering insights into the optimization process.

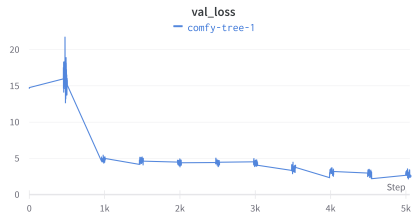


Figure 5.2: Validation Analysis for Contextualized PictoBERT: The x-axis signifies the training steps during different epochs, while the y-axis visually represents the validation loss.

The train loss initiates at a value of 16.5 at step 1 and maintains a range of around 16-20. Around step 500, it undergoes a significant drop from 21 to 5.5. Subsequently, it continues its descent, reaching a value of 2.86 at step 4999.

Conversely, the validation loss begins around 14.6 at step 0 and gradually decreases to approximately 4.7 around the 1k-step mark. It then exhibits oscillations, fluctuating between 5 and 4.7 for a brief period. Afterward, it sharply drops to approximately 4.2 around step 1400, followed by additional segments of oscillations within a certain range. Finally, it undergoes a steady linear decline, culminating at a value of 2.55 around step 5k.

The training results for Gloss PictoBERT are presented in Figure 5.3, which shows the progression of the training loss. Similarly, the fluctuation in the validation loss for Gloss PictoBERT is depicted in Figure 5.4.

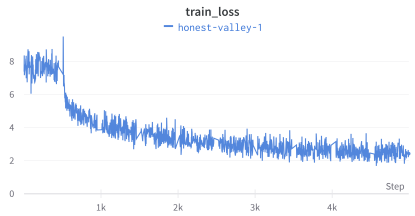


Figure 5.3: Training Progress of Gloss-Enhanced PictoBERT: the model's performance evolves during training, with the x-axis representing training steps across multiple epochs and the y-axis showcasing the corresponding loss values.

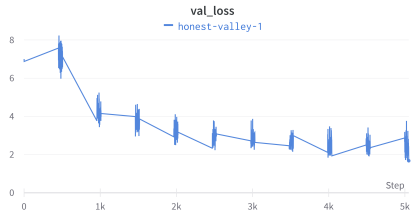


Figure 5.4: Validation Analysis for Gloss-Enhanced PictoBERT: The x-axis indicates training steps during different epochs, while the y-axis represents the validation loss.

The train loss originates at a value of 7.4 and exhibits oscillations within the range of 7.8 up to step 500. It then experiences a drop to 6 around step 520. Subsequently, it continues to oscillate but gradually decreases, ultimately reaching a value of 1.99 at step 5k. On the other hand, the validation loss commences at 7 at step 9, displaying oscillations in the range of 7.5 to 7.1. It then enters a linear decreasing phase with a notable drop to 3.7 at step 960. It continues this pattern, alternating between oscillations within a range and linear decline until it reaches a value of 1.6 at step 5k.

For fine-tuning, we provide additional information in Figure 5.5 and Figure 5.6.

These figures shed light on the fine-tuning process for Contextualized PictoBERT and Gloss PictoBERT.

During fine-tuning, PictoBERT contextualized exhibits a training loss that initiates at 1.2 and undergoes oscillations between 0 and 3 until step 17k, where it reaches a value of 1.9. In contrast, the validation loss starts at 2 and gradually decreases to 1.4 by step 17k.

In the fine-tuning phase, PictoBERT gloss training loss exhibits oscillations within the range of 0.5 to 4, starting from 2.5 and reaching 1.7 by step 18k. It reaches its maximum value of 3.9 in three distinct steps. On the other hand, the validation loss decreases from 0.9 to 0.8 by step 19k.

Figure 5.5: Contextualized Fine-Tuning of PictoBERT. The x-axis represents steps, while the y-axis shows the corresponding loss values.

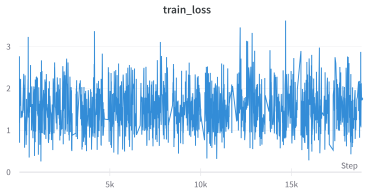


Figure 5.6: In this figure, we provide a detailed view of the fine-tuning process for Gloss PictoBERT. It displays the model's training progress, with the x-axis denoting training steps during various epochs and the y-axis presenting the loss values.

Type	Train Loss	Val Loss	Batch Size	Epochs	Time
CT train	2.71	2.55	112	10	50m
GL train	1.99	1.67	112	10	1h
CT ft	1.21	1.45	32	10	2h12m
GL ft	1.74	0.88	32	10	1h

Table 5.1: Training and Fine-Tuning Performance Metrics This table presents a comparative overview of training and fine-tuning performance metrics for two variants of PictoBERT: Gloss (GL) and Contextual (CT). It includes metrics such as training loss, validation loss, batch size, number of epochs, and time taken for each variant.

Complementing the visual representation of the loss trends, Table 5.1 presents a comprehensive overview of the loss values for each model, offering a detailed reference for further analysis.

Analyzing PictoBERT's Performance We delve into the evaluation of PictoBERT in comparison to n-grams LM on the test set, employing two distinct modes:

- Pre-trained;
- Fine-tuned over ARAASAC vocabulary.

Model Version	Cross-entropy	PPL
PictoBERT contextualized embeddings	2.523259	12.469173
PictoBERT gloss embeddings	3.133908	22.963542
n-gram (order=2)	5.506213	246.216936
n-gram (order=3)	4.340335	76.733222
n-gram (order=4)	3.951958	52.037178
n-gram (order=5)	3.888771	48.850834
n-gram (order=6)	3.887938	48.810126
n-gram (order=7)	3.894768	49.144648

Table 5.2: Pre-trained n-grams vs PictoBERT evaluated with PPL and Cross-entropy metrics over test set. Highlighted in green are the better values for each column.

top-1	top-9	top-18	top-25	top-36	PPL	Model version
0.351562	0.653061	0.732621	0.767060	0.804369	29.978450	Fine-tuned PictoBERT contextualized
0.337372	0.632972	0.711256	0.747768	0.786511	32.814500	Fine-tuned PictoBERT gloss
0.134942	0.434831	0.528973	0.576123	0.622162	78.350966	n-gram (order=2)
0.226703	0.484204	0.542785	0.565010	0.589617	24.272996	n-gram (order=3)
0.242896	0.443086	0.486744	0.503413	0.524210	17.333698	n-gram (order=4)
0.233688	0.424671	0.465788	0.481981	0.502302	16.470492	n-gram (order=5)

Table 5.3: Fine-tuned on ARASAAC mapping of n-grams vs PictoBERT evaluated with PPL and Top N accuracy over test set. Highlighted in green are the better values for each column.

Comparative Analysis The evaluation is carried out using models trained on the test dataset, comprising the following models:

- PictoBERT (contextualized embeddings);
- PictoBERT (gloss embeddings)
- n-gram order 2-7 (n-gram models)

The test file is `test_childes_all.pt` and the tokenizer path is `childes_all_new.json`. In Table 5.2, we observe that contextualized PictoBERT exhibits lower PPL and Cross-Entropy values. Notably, the experimental results for cross-entropy and PPL of pretrained n-gram LM models, when compared to contextualized PictoBERT and gloss models, align closely with the values reported in the paper [40] in their Table 2.

We compare the fine-tuned PictoBERT and n-gram models on the ARASAAC mapping using the test dataset, as presented in Table 5.3. Notably, the Top-N

accuracy values align closely with those reported in the paper [40], as indicated in Table 6. However, there are discrepancies in the PPL values, with contextualized PictoBERT registering a higher value of 10.4 compared to the paper, and other values showing an approximately 10% increase relative to their reported results.

Embarking on Interactive Pictogram Inference To put PictoBERT's practical utility on display, we provide an illustrative example of pictogram inference. Using the best performing model, PictoBERT contextualized, fine-tuned on ARASAAC mappings, Figure 5.7 showcases an interactive HTML interface, revealing the top 16 predicted pictograms for a given sentence.

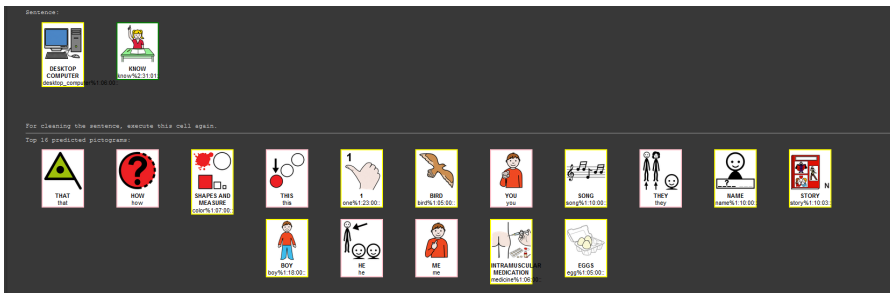


Figure 5.7: PictoBERT HTML interface. It employs an HTML interactive method to display the top 16 predicted pictograms corresponding to a given sentence.

5.3 PictoViLT

In this section, we dive into the intricate details of PictoViLT, fine-tuned with the masking technique L1, L2, and L3, designed to enhance its performance and capabilities. Our analysis is presented in a structured manner, providing a comprehensive overview of various aspects.

Logic 1 Based on the L1 logic, the training loss for Masking L1 over 2 epochs is meticulously visualized in Figure 5.8.

It is crucial to note that the final training loss attains an impressive value of 0.001366, underscoring the model's effectiveness in optimizing its understanding of the data.

Understanding resource utilization is essential to assess the feasibility and efficiency of any model. In the case of PictoViLT with Masking L1, we observe a memory consumption of 4823.625 MB, crucial for scalability and deployment considerations. The overall trend of memory consumption is thoughtfully illustrated in Figure 5.9, providing insight into memory dynamics throughout the training process.

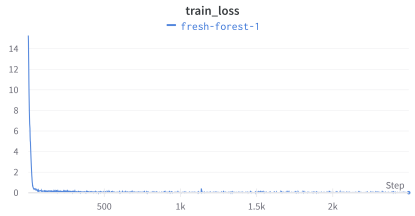


Figure 5.8: This figure provides a comprehensive overview of the training loss during the L1 fine-tuning process of the PictoViLT model. The plot visualizes the progressive reduction in training loss over 2 epochs, reflecting the model's ability to optimize its understanding of the data.

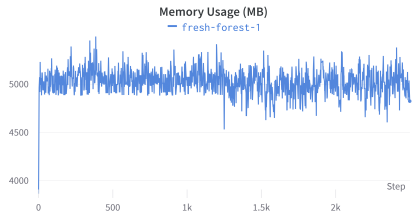


Figure 5.9: The chart captures the dynamic allocation of memory resources, offering insights into the model's memory utilization trends throughout the training process.

GPU utilization is another critical facet, especially in contemporary DL applications. To this end, we meticulously monitor and present GPU-related metrics. Figures 5.10 and 5.11 provide a clear representation of GPU power usage in watts and GPU temperature in degrees Celsius, respectively. These metrics shed light on the computational demands and thermal characteristics

of PictoViLT during Masking L1 fine-tuning.

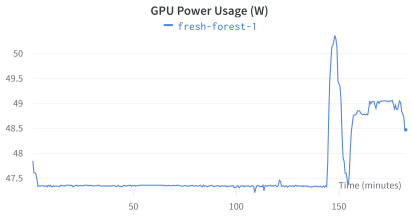


Figure 5.10: This figure delves into the GPU power consumption during the L1 fine-tuning of PictoViLT. By visualizing the power usage in watts, we gain a deeper understanding of the computational demands imposed by the model, which is crucial for resource management and optimization.

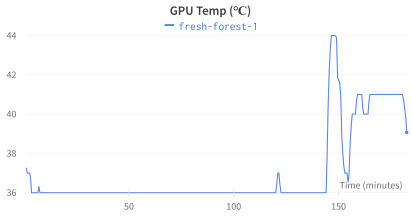


Figure 5.11: Thermal management is a critical aspect of model training, and this figure sheds light on the GPU temperature dynamics during the L1 fine-tuning process. Monitoring temperature in degrees Celsius provides essential insights into the model's thermal behavior and potential cooling requirements.

During fine-tuning of PictoViLT's Logic 1, the training loss starts around 15 at step 0, steadily decreasing to approximately 0 by step 3k. This descent includes a rapid drop around step 100. GPU memory consumption remains stable at around 5000 MB throughout, with GPU power usage fluctuating between 47W and 50W and an average temperature of 40°C.

Logic 2 We embark on a detailed exploration of the fine-tuning process for the PictoViLT model using L2 masking. Figure 5.12 serves as our guide, showcasing how the model's training loss evolves over a span of 2 enlightening epochs. Figure 5.13 offers us a glimpse into the art of memory utilization by the PictoViLT model during L2 masking fine-tuning. It provides a vivid picture of how the model strategically allocates and manages memory resources. The last loss is 0.02871, memory 8055.371 MB in Figure 5.13. GPU power usage in W in Figure5.14 while temperature values in the chart of Figure 5.15.

Figure 5.14 serves as our compass, tracking the power usage of the GPU

Figure 5.12: This figure presents the training progress of the PictoViLT model during L2 masking fine-tuning. It illustrates how the training loss evolves over a span of 2 epochs, reflecting the model's adaptation and learning process during this phase.

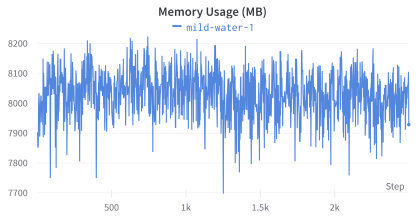


Figure 5.13: Here, we visualize the memory consumption of the PictoViLT model throughout the L2 masking fine-tuning. The graph provides information on how memory resources are allocated and managed during the training process.

throughout the L2 masking fine-tuning process. Figure 5.15 acts as our thermometer, recording the GPU's temperature in degrees Celsius during L2 masking fine-tuning. These thermal cues provide a deeper understanding of how the model's training impacts the GPU's thermal dynamics.

Similarly, in the PictoViLT fine-tuning for Logic 2, the training loss fol-

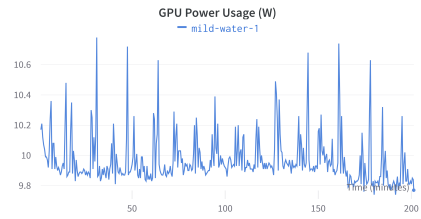


Figure 5.14: This figure tracks the power usage of the GPU during the L2 masking fine-tuning process. By monitoring power consumption in watts, we gain an understanding of the computational demands placed on the GPU

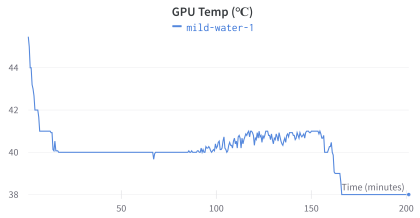


Figure 5.15: his figure displays the GPU's temperature in degrees Celsius throughout L2 masking fine-tuning, providing insights into the model's thermal behavior.

lows a consistent descent to approximately 0. Memory consumption averages around 8100 MB, with an average temperature of 40°C, and power usage hovers around 10W.

Logic 3 Here, we embark on a captivating exploration of the PictoViLT model's L3 masking fine-tuning.

Our journey begins with Figure 5.16, a visual representation of the model's training progress during L3 masking fine-tuning. This chart illustrates the evolution of the training loss over 2 insightful epochs. In particular, the final training loss stands at 0.05948, reflecting the model's adaptation during this phase. Figure 5.17 offers a glimpse into the PictoViLT model's memory consumption during L3 masking fine-tuning. The graph provides insights into how memory resources are allocated and managed, shedding light on the model's memory prowess. Memory consumption is 7556.656 MB.

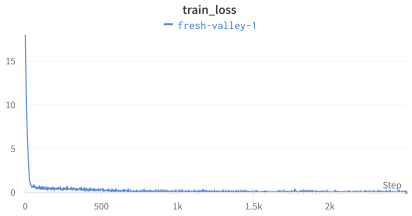


Figure 5.16: L3 fine-tuning training loss. It showcases how the training loss evolves over a span of 2 insightful epochs, providing valuable insights into the model's adaptation and learning process during this phase.

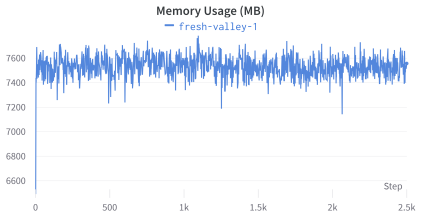


Figure 5.17: L3 fine-tuning memory consumption. It provides insights into allocating and managing memory resources, shedding light on the model's memory prowess.

Figures 5.18 and 5.19 act as our guides, tracking the GPU's power usage in watts and its temperature in degrees Celsius throughout L3 masking fine-tuning. These metrics offer valuable insights into the model's computational demands and thermal dynamics during this phase.

In the PictoViLT fine-tuning for Logic 3, the training loss exhibits a gradual decline, reaching approximately 0, mirroring the patterns observed in Logic 1 and Logic 2. Memory utilization averages around 7600 MB, while GPU

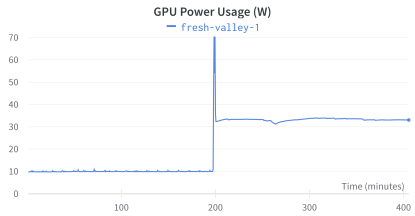


Figure 5.18: L3 fine-tuning GPU power usage. By monitoring power consumption, we gain a deeper understanding of the computational demands placed on the GPU.

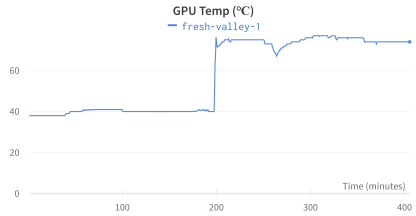


Figure 5.19: L3 fine-tuning GPU temperature in C° chart providing valuable information about the model's thermal dynamics during this phase.

Label	Train Loss	Memory Usage
Logic 1	0.0014	4,823 MB
Logic 2	0.0287	8,055 MB
Logic 3	0.0595	7,556 MB

Table 5.4: Results of metrics for logical models. It is a comprehensive overview of the performance metrics for the three logical models. These metrics include training loss and memory usage for each logic. The table serves as a valuable reference point to assess the effectiveness of these models.

power usage fluctuates between 10W and 40W, with GPU temperature averaging 60°C.

Comparisons The performance metrics for the three logical models are summarized in Table 5.4 below. 2 While Logic 1 demonstrates impressive training loss and memory management, Logic 2 and Logic 3 exhibit slightly higher training losses, with optimized memory management.

During the fine-tuning process for Logic 1, we observed a significant reduction in the training loss, which decreased from an initial value of 14.28 to a final value of 0.018, see Figure 5.20. This training phase lasted approximately 10 hours and 25 minutes, comprising 3,000 steps, equivalent to approximately 40 minutes per epoch.

In contrast, in Figure 5.22, the validation loss demonstrated a more modest

improvement, decreasing from 0.024 to 0.021, with a corresponding accuracy rate of approximately 35%.

These computations were equipped with 16GB of memory, see Figure 5.21, and the entire fine-tuning process for L1 took approximately 52 minutes.

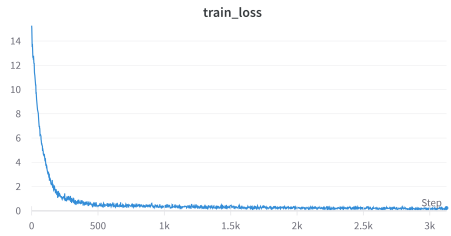


Figure 5.20: L1 Train loss.

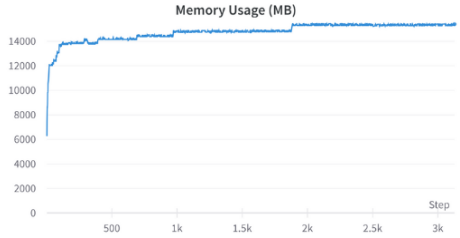


Figure 5.21: L1 Memory usage.

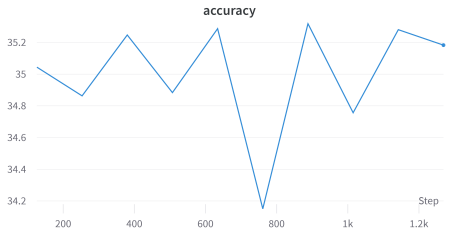


Figure 5.22: L1 Accuracy over validation.

In the training phase for Logic 2 (L2), we executed a total of 2,000 steps, with the entire process taking approximately 16 hours to complete. During this time, the training loss exhibited a significant reduction, decreasing from an initial value of 15.11 to a final value of 0.0138, see Figure 5.23.

However, in contrast to the training loss, the validation loss showed a slight increase, moving from 0.03 to 0.04 in Figure 5.25, along with an accuracy rate of 39%.

Remarkably, the validation process consumed a substantial amount of memory, approximately 88GB in Figure 5.24, yet it was completed efficiently in approximately 1 hour and 40 minutes.

In the case of Logic 3 (L3), the training phase involved a total of 2,000 steps, spanning a period of 19 hours. During this extensive training session,

Figure 5.23: L2 Train loss.

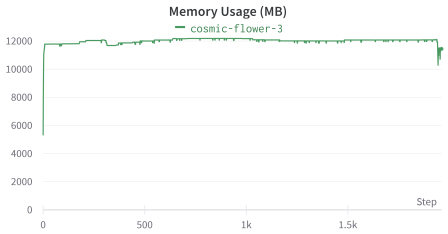


Figure 5.24: L2 Memory usage.

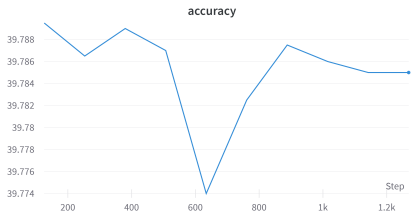


Figure 5.25: L2 Accuracy over validation.

the training loss experienced a substantial drop, dropping from an initial value of 15.2 to a final value of 0.12. The training for Logic 3 was conducted on an NVIDIA GEFORCE 3090 graphics card, equipped with 13GB of RAM.
On the other hand, the validation loss exhibited an increase from 0.1 to 0.3, resulting in an accuracy rate of approximately 37.8% as in Figure 5.26.
Notably, the validation process required a significant amount of memory, approximately 112GB, and took approximately 2 hours and 40 minutes to complete.

Figure 5.26: L3 Accuracy over validation.

The results in Table 5.5 offer insight into PictoViLT's performance under different fine-tuning logics. In L1, the model achieved efficient training but showed room for improvement in validation metrics. L2 showed a notable

Train Loss	Time	Memory	Steps	Val Loss	Accuracy	Val Memory
0.02	10h	15GB	3k	0.02	35%	16GB
0.01	16h	11GB	2k	0.04	39%	88GB
0.12	19h	13GB	2k	0.32	38%	112GB

Table 5.5: The table presents the results of on-server fine-tuning for PictoViLT. The table summarizes key metrics, including train loss, training time, memory usage during training, number of steps, validation loss, accuracy, and validation memory consumption

reduction in train loss but a slight increase in validation loss, while L3 had the longest training duration and exhibited a significant increase in validation loss.

Inference In this section, we explore the results of top-k inference using PictoViLT under three different fine-tuning logics: Logic 1, Logic 2, and Logic 3.

Figure 5.27: Baby pictogram.

Table 5.6 presents the top-10 word predictions generated by PictoViLT for the baby pictogram, in Figure 5.27 under each logic.

For the first row of predictions, Table 5.7 displays the normalized probability distributions (softmax) of model output scores.

After fine-tuning on the server, the inferences are these in Table 5.8.

Logics	Top-10
1	look, baby, horse, bear, man, pig, dog, cow, calf, puppy
2	city, baby, shoo,t look, move, drink, character, job, toy
3	day, job, baby, tub, bathroom, spotlight, sheep, pet, stall

Table 5.6: Top-10 word predictions for the baby pictogram under three fine-tuning logics.

Token	Normalized prob.
look	0.1058363
baby	0.1058284
horse	0.1057948
bear	0.1057407
man	0.1057398
pig	0.1055579
dog	0.1054733
cow	0.1042075
calf	0.0822827
puppy	0.0735386

Table 5.7: Normalized probabilities for each word in the sequence.

Logics	Top-10
1	man, person, boy, child, character, cartoon, drawing, baby, dog, symbol
2	baby, body, look, seal, swim, move, puppy, bone, bath, litter
3	float, bone, swim, tattoo, baby, move, bath, tub, canoe, knee

Table 5.8: Updated top-10 word predictions for the baby pictogram after server fine-tuning.

For the sentence ”it is bread cut with [MASK]” representing a cutting bread

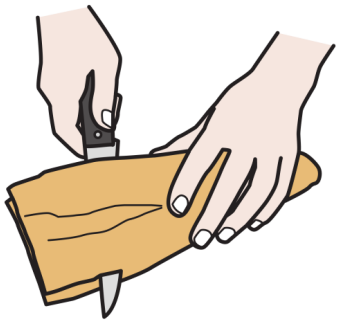


Figure 5.28: Bread pictogram.

pictogram, the inferences are detailed in Table 5.9.

”You can use this [MASK].” is the starting sentence on the ketchup pictogram in Figure 5.29, the inferences are detailed in Table 5.10.

All of those token predictions ought to be evaluated having the awareness

Logics	Top-10
1	paper, hands, hand, pieces, butter, fingers, scissors, finger, slices, knife
2	bone, dough, butter, cut, fry, peel, pieces, knife, garlic, cuts
3	skin, butter, dough, pepper, hand, flour, sugar, salt, knife, garlic

Table 5.9: Top-10 word predictions for the cutting bread pictogram sentence after server fine-tuning.

Figure 5.29: The ketchup pictogram.

Logics	Top-10
1	product, one, bottle, color, bag, icon, item, label, water, bottles
2	bag, cover, season, cap, bottle, label, lid, pouch, oil, salt
3	stuff, bag, jar, honey, oil, water, top, pepper, drink, bottle

Table 5.10: Top-10 word predictions for the bottle of ketchup sentence after server fine-tuning.

that the word ”ketchup” is not included in the dataset.

The top-k inference results for the baby pictogram demonstrate variations in word predictions under different fine-tuning logics. L1, L2, and L3 logics produce distinct predictions, highlighting the impact of fine-tuning strategies on model behavior. These results provide insights into the model’s adaptability and potential suitability for specific tasks based on the chosen fine-tuning approach.

When fine-tuned on the Conceptual Captions dataset, PictoViLT was tested with the baby pictogram, resulting in Table 5.11. Tried PictoViLT fine-tuned on Conceptual Captions dataset to infer on baby pictogram, the table shows

Logics	Top-10
1	run, dog, jump, exercise, goat, template, swimmer, wet, puddle, snail
2	letter, photograph, bench, teenager, movie, balance, document, temple, fold, code
3	seal, stamp, tattoo, baby, sleep, wardrobe, wheelchair, balance, bob

Table 5.11: Inference results for the baby pictogram using PictoViLT fine-tuned on the Conceptual Captions dataset.

the worst results respect to CommonGen dataset.

Notably, when fine-tuned on the Conceptual Captions dataset, the model's performance on the baby pictogram differs from that on the CommonGen dataset.

Interpretability This section discusses the interpretability of the PictoVILT model, which seamlessly integrates text and image processing. This integration enables the model to generate predictions for [MASK] tokens within text, while simultaneously highlighting relevant areas within an associated image.

The interpretability process involves several key steps. A method calculates text and image embeddings using the ViLT model, two types of embeddings, textual and visual, are then combined with attention masks to identify which parts of the text and images are important to the model.

To discern the crucial connections between text and images, attention masks are employed. These masks reveal which parts of the text and images hold significance for the model's understanding.

The distances between text and images are computed. A variant of Earth Mover's Distance is utilized to compute a transport matrix that outlines how text tokens correspond to image patches. This matrix is instrumental in establishing relationships between text and images during the inference process.

The images are resized to dimensions suitable for processing by the model, ensuring that their original appearance remains intact.

A cosine distance matrix is computed to measure the similarity between

embedding vectors, particularly between text and image embeddings. This matrix helps in assessing the degree of similarity between two vectors.

During inference, the ViLT model completes the [MASK] tokens within text. At the same time, it generates an image with highlighted regions based on the associations between text and images. This highlighting provides valuable insights into how the model perceives and interprets the content.

During inference, an initial image is opened, scaled to the size required by the model, and used as input. The VILT model is then used to infer the [MASK] token into the text, completing it. Next, the highlight area of the resulting image is calculated based on the relationship between the text and the images. To evaluate all logic's attention behavior, the ViLT model is switched between each of them.
Masking logits are extracted from the model output taking into account only text features. Logits are transformed into probabilities through a softmax, obtaining the probability values for each word of the vocabulary. The token with the highest probability is selected and its index is substituted into the text encoding.

The interpretability interface, shown in Figure 5.30, is realized using the Gradio open source library to create ML applications. [5] The framework is applied to analyze the ViLT-based model's behavior when given a pictogram and a caption containing [MASK] tokens to be filled, see Figure 5.31. The application visually demonstrates where the model's attention is focused when predicting a specific word in the caption. This insight enhances our understanding of the model's decision-making process and its ability to connect textual and visual information.

To gain insight into how attention operates within the three masking logic

[5] Gradio

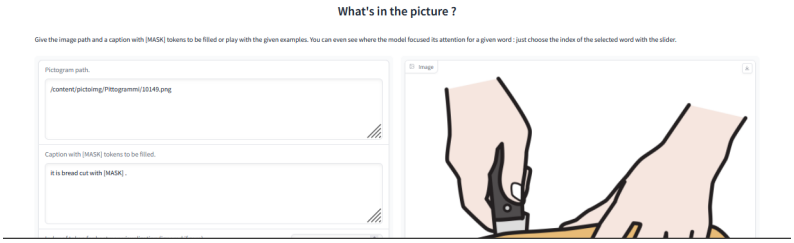


Figure 5.30: Gradio interface. There is a text box for the input-masked sentence, a text box for the word we want to focus on, and an image box for the pictogram input. It shows the predicted token for MASK and the image with highlighted patches as output.

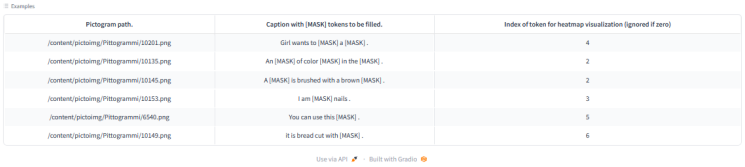


Figure 5.31: Gradio interface: default pictograms table ready to click inference examples.

models, we start our analysis with a ketchup pictogram, as shown in Figure 5.29.

Starting with Logic 1, as shown in Figure 5.32, the model focuses its attention on the word ”bottle”, highlighting the contours of the bottle itself, as demonstrated in Figure 5.33. Notably, this attention is precise, even when examining the ”tch” token, as seen in Figure 5.35.

Moving to Logic 2, illustrated in Figure 5.36, the model’s attention is drawn to the predicted token ”bottle”. However, the focus here shifts more towards the label, as evident in Figure 5.37, compared to the L1 example. For the ”ke” token, as depicted in Figure 5.38, L2’s attention shifts away from the bottle’s shape, narrowing down to the central portion, including the label (Figure 5.39).

Now, examining Logic 3 in Figure 5.40, the model’s attention centers on the

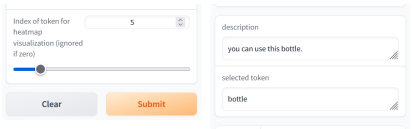


Figure 5.32: L1 visualization app for ketchup pictogram.

Figure 5.33: L1 attention on ketchup pictogram.

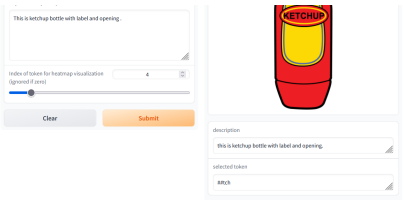


Figure 5.34: L1 visualization app for ketchup pictogram on ”tch” token.

Figure 5.35: L1 attention on ”tch” of ketchup pictogram.

predicted ”ke” token. However, it sharply focuses on the first two letters of the bottle label, as shown in Figure 5.41, omitting the bottle’s contour and the rest of the label, a notable shift from the L2 example.

Let’s explore the model’s behavior further using a pictogram of a girl, as displayed in Figure 5.42.

In Figure 5.44, when examining Logic 1, the model’s attention gravitates

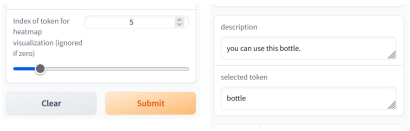


Figure 5.36: L2 visualization app for ketchup pictogram.

Figure 5.37: L2 attention on predicted ”bottle” token of ketchup pictogram.

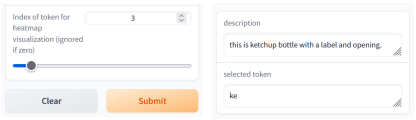


Figure 5.38: L2 visualization app for ketchup pictogram.

Figure 5.39: L2 attention on predicted ”ke” token of ketchup pictogram.

towards the word ”wants” emphasizing the girl’s eyes. For Logic 2, as illustrated in Figure 5.46, the model’s attention on ”bite” corresponds to the girl’s mouth expression, with candy in front.5.46.

Interestingly, Logic 1 and Logic 3 produce the same prediction for this sentence, as seen in Figure 5.47. However, the attention patterns differ significantly between the two. In Figure 5.48, L1 highlights the facial shape and tongue, while L3 focuses on the eyes and tongue, as depicted in Figure 5.49.

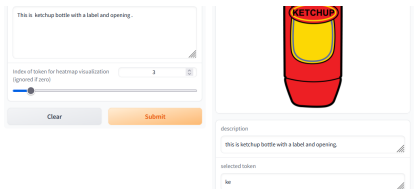


Figure 5.40: L3 visualization app for ketchup pictogram.

Figure 5.41: L3 attention on predicted ”ke” token of ketchup pictogram.

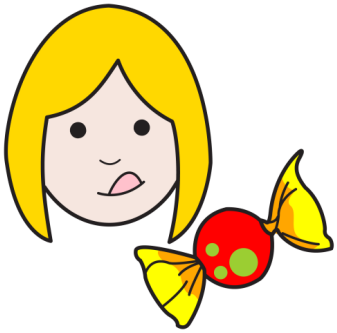


Figure 5.42: Girl pictogram.

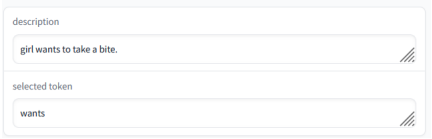


Figure 5.43: L1 visualization app for girl pictogram.

Figure 5.44: L1 attention on predicted ”wants” token of girl pictogram.

[htbp]

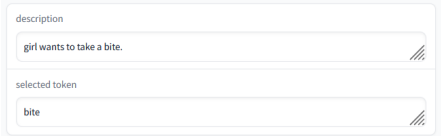


Figure 5.45: L2 visualization app for girl pictogram.

Figure 5.46: L2 attention on predicted ”bite” token of girl pictogram.

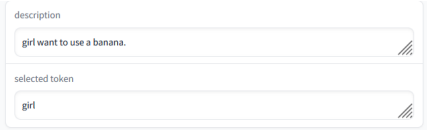


Figure 5.47: L1 - L3 Visualization app for girl pictogram.

Figure 5.48: L1 attention on predicted ”girl” token of girl pictogram.

Figure 5.49: L3 attention on predicted ”girl” token of girl pictogram.

Lastly, let us delve into the sentence ”It is bread cut with [MASK].” corresponding to the pictogram in Figure 5.28.

In the L2 model, as shown in Figure 5.50, the attention is directed towards the word ”cut”, particularly focusing on the patches of the knife, especially its tip, as illustrated in Figure 5.51. However, for the word ”bread”, as seen in Figure 5.52, L2’s attention shifts toward the brown parts of the bread, as depicted in Figure 5.53.

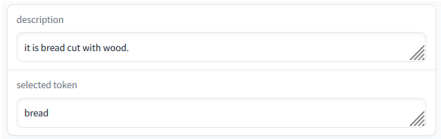


Figure 5.50: L2 visualization app for bread pictogram.

Figure 5.51: L2 attention on predicted ”cut” token of bread pictogram

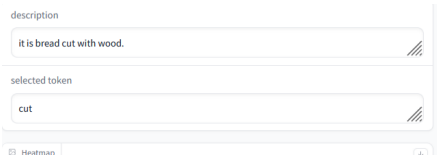


Figure 5.52: L2 visualization app for bread pictogram.

Figure 5.53: L2 attention on predicted ”bread” token of bread pictogram.

However, in Figure 5.52, the ”bread” attention of the L2 model is focused on the brown parts of the bread in Figure 5.53.

5.4 Comparisons

In this section, we compare in detail PictoBERT and other models using the test set. Before conducting these comparisons, PictoBERT underwent specific modifications to adapt to the CommonGen dataset, known for its word-sense-based challenges. These adaptations are described below.

Evaluating PictoViLT: Unveiling its Performance Metrics To fully assess the capabilities of PictoViLT, a meticulous evaluation strategy has been devised. The primary objective is to gauge its performance in terms of both Top-N accuracy and PPL, see Section 5.1.

A fundamental aspect of the evaluation process revolves around the measurement of accuracy, initializing counters to zero to record correct predictions at each Top-N level.

At the outset, a meticulous examination of the test dataset is conducted. This scrutiny includes keeping tabs on the total number of examples contained within the dataset. Furthermore, a crucial variable known as ”total loss” is monitored. This variable serves as an aggregation point for the losses computed by the model across all test examples.

The evaluation process is a meticulous and iterative endeavor. It works by iterating through the test `dataloader`, which provides batches of test data. Each batch, a treasure trove of insights, consists of essential components: input IDs, an attention mask to spotlight pertinent tokens, target labels for model evaluation, and pixel values that encode the associated images.

At the heart of the evaluation lies the execution of a forward pass through the PictoViLT model. Fueled by the input data, this forward pass unfolds the model’s predictive prowess, culminating in a set of predicted logs.

A pivotal step in the evaluation process is accumulating the total loss. This aggregation involves the summation of losses stemming from each test batch, painting a comprehensive picture of the model's performance.

The core of evaluation hinges on the model's predictions. These predictions, derived by applying the softmax function to the model's logits, represent the estimated probabilities for each class. To scrutinize and dissect these predictions, a masked position mask is meticulously crafted. It identifies the positions within the input where the tokens were originally masked.

With predictions in hand, the next step is to harness the power of Top-N predictions. For each predefined Top-N value, the model undertakes the task of computing predictions. This feat is accomplished through the utilization of the torch top-k function, meticulously applied to the logits.

With the Top-N predictions in tow, a meticulous comparison unfolds. The focus is on the masked positions within the input. For each example residing within the batch, if at least one of the Top-N predictions aligns with the actual label at a masked location, it is unequivocally counted as a correct prediction for the corresponding Top-N value.

The assessment does not stop at accuracy alone; it delves deeper. PPL is computed as the exponential of the average total loss divided by the total number of test examples; PPL provides valuable insights into the model's predictive nuances.

The results of this comprehensive evaluation are meticulously compiled into a structured dictionary format. This record encapsulates vital data, including PPL and accuracy percentages for each predefined Top-N value.

top-1	top-9	top-18	top-25	top-36	PPL	Parameters	Model version
0.906977	0.953488	0.953488	0.976744	1.000000	1.001929	135M	PictoViLT Logic 1
0.680000	0.700000	0.700000	0.700000	0.700000	1.002402	135M	PictoViLT Logic 2
0.920000	0.960000	0.980000	0.980000	0.980000	1.026120	135M	PictoViLT Logic 3
0.351562	0.653061	0.732621	0.767060	0.804369	29.978450	312M	Fine-tuned PictoBERT contextualized
0.337372	0.632972	0.711256	0.747768	0.786511	32.814500	312M	Fine-tuned PictoBERT gloss
0.065407	0.414669	0.522940	0.582632	0.646611	267.772432	4.7M byte	n-gram (order=2)
0.254961	0.638355	0.740117	0.784093	0.821718	87.607856	4.7M byte	n-gram (order=3)
0.368630	0.726465	0.803937	0.838387	0.872837	55.007657	4.7M byte	n-gram (order=4)
0.406731	0.742499	0.812351	0.845214	0.876965	48.823797	4.7M byte	n-gram (order=5)

Table 5.12: Performance metrics comparison. Better results for each metric are highlighted in green color.

To furnish an organized view of the evaluation outcomes, the results dictionary undergoes a transformation into a Pandas `DataFrame`—an organized tableau that succinctly presents the findings.

The crux of this evaluation endeavor centers on the three-logic models. These models undergo scrutiny across an array of Top-N values, including **top-1**, **top-9**, **top-18**, **top-25**, and **top-36** accuracy, as well as PPL.

Table comparisons: Analysis of metrics To offer a concise yet comprehensive view of the evaluation results, we present the findings in Table 5.12.

The Table's rows are dedicated to contrasting the performance of various models across these metrics. Each row represents a distinct model variant under evaluation, and the corresponding metrics provide insights into their capabilities.

The PictoViLT 3 Logics model's performances are evaluated over a batch size of 8 and epochs 2.

Logic 1 impresses with its top-1 accuracy, soaring to 90.70%, showcasing its ability to make precise top-ranked predictions. Its PPL hovers around

top-1	top-9	top-18	top-25	top-36	PPL	Model version
0.978261	1.00	1.0	1.00	1.00	1.002157	Logic 1
0.580000	0.70	0.7	0.72	0.74	1.004225	Logic 2
0.500000	0.72	0.8	0.82	0.84	1.040458	Logic 3

Table 5.13: Evaluation metrics comparison table with 16 batch sizes and 10 epochs fine-tuning. In green color are the better values for each column.

1.00, reflecting its clarity in predictions. Logic 2 exhibits balanced performance across various Top-N values, maintaining competitive accuracy. Its PPL, although slightly above 1.00, demonstrates its overall coherence. Logic 3 emerges as a formidable contender, boasting high accuracy figures across Top-N values. Its PPL, while marginally higher, reinforces its capability.

Fine-tuned PictoBERT Contextualized presents moderate accuracy and a relatively high PPL, indicating its sensitivity to context. Fine-tuned PictoBERT Gloss aligns closely with contextualized PictoBERT, showing similar accuracy patterns and a marginally higher PPL.
N-gram (Order=2) shows limited accuracy and higher PPL, while n-gram (Order=3) improves accuracy and lowers PPL. n-gram (Order=4) enhances accuracy further, and n-gram (Order=5) culminates with improved accuracy and respectable PPL.

The model evaluations are further segmented into distinct training scenarios. Two such scenarios, captured in Table 5.13 and Table 5.14, provide insights into the impact of batch size and epochs on model performance.

Top-N accuracy and PPL comparisons of 3 Logics models with RGB image conversion fine-tuned with 16 batch sizes and over 10 epochs, see Table 5.13.

Top-N accuracy and PPL comparisons of 3 Logics models with RGB image conversion fine-tuned with 32 batch size and over 2 epochs, see Table 5.14.

top-1	top-9	top-18	top-25	top-36	PPL	Model version
0.9375	1.00	1.00	1.00	1.00	1.000316	Logic 1
0.6200	0.74	0.74	0.74	0.74	1.001896	Logic 2
0.5000	0.72	0.78	0.84	0.84	1.017408	Logic 3

Table 5.14: Evaluation metrics comparison table with 32 batch sizes and 2 epochs fine-tuning. Green values are the best for each metric reported.

top-1	top-9	top-18	top-25	top-36	PPL	Model version
0.954545	1.00	1.00	1.00	1.00	1.000402	Logic 1
0.880000	0.88	0.88	0.88	0.88	1.000309	Logic 2
0.940000	0.96	0.98	0.98	0.98	1.004670	Logic 3

Table 5.15: Evaluation metrics comparison table with 32 batch sizes and 8 epochs fine-tuning. Better results in green.

Model	top-1 incr.	top-9 incr.	top-18 incr.	top-25 incr.	top-36 incr.	PPL decr.
Logic 1	0.602983	0.346939	0.267379	0.232940	0.195631	28.978048
Logic 2	0.528438	0.226939	0.147379	0.112940	0.075631	28.978141
Logic 3	0.588438	0.306939	0.247379	0.212940	0.175631	28.973780

Table 5.16: Table reports improvements for each Top-N accuracy obtained by all 3 logics variants of PictoViLT respect to PictoBERT contextualized. Better values for each column are highlighted in green.

Transitioning to the domain of 32 batch sizes and 2 epochs (Table 5.14), Logic 1 consistently achieves impressive Top-N accuracy, underlining its robustness. Logic 2 maintains stability in accuracy, with a notable boost in top-25 and top-36 performance. Logic 3 shows remarkable strides in accuracy, especially in top-25 and top-36 categories. PPL remains reasonably steady, reaffirming the models' adaptability.

In addition, when comparing the results with 8 epochs of fine-tuning (Table 5.15), Logic 1's Top-N accuracy remains consistent. Logic 2 maintains its stability across all metrics, while Logic 3 continues to show remarkable progress in top-25 and top-36 accuracy. These findings suggest that longer fine-tuning may have a positive impact on the models' performance

To summarize, better performance is for batch size 32 and 8 epochs. So, in the following Table 5.16 the analysis of the improvement of each logic in

Model	Mean Improvement
Logic 1	0.329374
Logic 2	0.218265
Logic 3	0.306265

Table 5.17: Table reports the improvement mean obtained by all 3 logics variants of PictoViLT with respect to PictoBERT contextualized over the Top-N accuracy. The green line contains the best result.

Sentence	Masked
be build rail.	be build rail [MASK]
be life throw pillow	be life throw [MASK]
shoot leave on young branch move wind	shoot leave on young branch move [MASK]
how to bedroom grey wall	how to bedroom grey [MASK]
church morning break over lake	church morning break over [MASK]
beautiful couple cabin	beautiful couple [MASK]

Table 5.18: Table reports test dataset sentences where the only last token is masked to make a comparison between PictoViLT and PictoBERT for the ”Next pictogram prediction” task.

Top-N accuracy with respect to PictoBERT contextualized.

Table 5.17, representing the mean of improvement given by the 3 PictoViLT’s logics with respect to PictoBERT contextualized over Top-N accuracy, shows that Logic 1 reports a higher improvement followed by Logic 2.

Based on the average of accuracy metrics, Logic 1 surpasses the state of the art by an improvement of approximately 0.33, while when evaluated on top-1 accuracy, it outperforms it by a more significant margin of approximately 0.60.

Considering that the starting point task is the ”Next Pictogram prediction”, to make a more coherent comparison between PictoBERT and our solution, We tested top-N accuracy and PPL of PictoViLT logic 1 (the best) over the test set where always last token is masked. In Table below 5.18 an example of test dataset sentences where the last pictogram is masked.

To evaluate the improvement of our solution in the next sentence prediction task, we evaluate our best PictoViLT model Logic 1 over a test set where the

Model	top-1 incr.	top-9 incr.	top-18 incr.	top-25 incr.	top-36 incr.	PPL decr.
Logic 1	0.977273	1.0	1.0	1.0	1.0	1.000478

Table 5.19: Table reports improvements for each Top-N accuracy obtained by PictoViLT Logic 1 over the last token masked sentences test set demonstrating better performance for the next sentences prediction task.

last token's sentence is masked. The results of evaluation metrics are in Table 5.19.

Conclusions

This thesis presented the process of design, development, and analysis of the results of a fine-tuned VL transformer model PictoViLT with 3 different masking logics, from single token to multi-tokens to outperform PictoBERT in the next sentence prediction task. We started from the study of PictoBERT in its two embedding versions and the training and fine-tuning were re-performed on the dataset adapted to the ARASAAC mappings.

Starting from the evaluation metrics' values, Top-N accuracy and PPL, of the model compared to the n-gram statistical LM, a solution was developed that exploits a text dataset associated with the ARASAAC pictograms but this time abandoning the word-sense, strength of PictoBERT.

The results obtained demonstrate an average improvement in accuracy values of **+0.30** for all 3 logics while logic 1, with masking of only one token per sentence, stands out for bringing a **+0.60** improvement on top-1 accuracy.

PictoViLT demonstrates that the transition to multimodal strengthens the ability to understand the concept model thanks to semantic text-image associations. We wanted to delve deeper into how the model focused attention on different areas of the image based on the selected words of the associated sentence, verifying the interpretability of the model.

Models were then tested for making different inferences from a masked

sentence. The models were run on Colab and servers with different combinations of batch sizes and epochs demonstrating the best results with a greater number of epochs (8) and a larger batch size (32).

The objective achieved is part of the AAC, where integrated with a mobile application becomes a fundamental support for people with CCN. The integrated framework has the potential to change the social participation of these people by simplifying their communication with others and speeding up learning.

5.5 Limitations and Future Improvements

Compared to single-modal AI, the integration of several sources of data on which models are trained results in a more comprehensive understanding of the context and a more accurate prediction.

5.5.1 Limitations

Dataset Constraints Despite that, one limitation pertains to the datasets containing words related to ARASAAC pictograms, which may be restricted.

Symbolic Limitations Due to the need to adhere to conventional symbols, pictograms may, in certain contexts, be less illustrative and less effective in representing complex concepts. Furthermore, the same pictograms exhibit limitations in terms of explicability.
It could be essential to develop techniques that enable the creation of more specific visual representations, such as conjunction or personalized pictograms.

5.5.2 Future Improvements

Technical Advancements Future research may look at ways to develop WordNet graphs to connect concepts within sentences by employing Graph Neural Networks like Edge Aggregated Graph Attention Networks for integrated representations.

ViLT might be improved by adding a cross-attention layer to integrate such representations with text and pictograms.

Integration into Applications and Community Engagement As a future development, the predictive models explored in this thesis, particularly PictoVILT, can be seamlessly integrated into an AAC system application. An example of an existing AAC application is reported in Figure 5.54. [6]

Figure 5.54: Screenshots of the **Leeloo** AAC app acquired from the Play Store. App for kids with a carefully crafted design with the ability to read the phrases generated by a Text-To-Speech system which allows multiple voices' selection.

Integration of AI models, such as PictoVILT, into the application can be seen as the core AI logic that operates within the framework. The use of an

[6] Leeloo

integrated framework provides the flexibility to seamlessly introduce new features or modifications, thereby enhancing its educational capabilities and enabling adaptation to individual user requirements. Furthermore, envisioning a forthcoming release of an integrated application, we foresee the potential establishment of a thriving user community. This community would play a vital role by providing feedback on existing functionalities and, even more importantly, by contributing innovative ideas and recommendations for potential features to implement. Such a collaborative community would significantly benefit from a similar open-source platform.

With continuous support and the implementation of novel and effective functionalities, this integrated framework has the potential to become a recognized and widely used tool within the community dedicated to the education of individuals facing communication difficulties.

Bibliography

[1] O. Agarwal, H. Ge, S. Shakeri, and R. Al-Rfou. Large scale knowledge graph based synthetic corpus generation for knowledge-enhanced language model pre-training, 2020. arXiv: 2010.12688 [cs.CL].

[2] I. Almeida, A. Moreira, and J. Ribeiro. High-tech augmentative and alternative communication devices: observing childrenrsquo;s need for help and interaction with caregivers. *Social Sciences*, 12(5), 2023. issn: 2076-0760. doi: 10.3390/socsci12050310. url: https://www.mdpi.com/2076-0760/12/5/310.

[3] N. R. Andzik, J. M. Schaefer, and V. L. Christensen. The effects of teacher-delivered behavior skills training on paraeducators' use of a communication intervention for a student with autism who uses aac. *Augmentative and Alternative Communication*, 37(1):1–13, 2021. doi: 10.1080/07434618.2021.1881823.

[4] R. M. Barker, S. Akaba, N. C. Brady, and K. Thiemann-Bourque. Support for AAC use in preschool, and growth in language skills, for young children with developmental disabilities. *Augmentative and Alternative Communication*, 29(4):334–346, 2013. doi: 10.3109/07434618.2013.848933.

[5] W. Berrios, G. Mittal, T. Thrush, D. Kiela, and A. Singh. Towards language models that can see: computer vision through the lens of natural language, 2023. arXiv: 2306.16410 [cs.CL].

[6] F. Chiarotti and A. Venerosi. Epidemiology of autism spectrum disorders: a review of worldwide prevalence estimates since 2014. *Brain Sciences*, 10(5):274, May 2020. doi: 10.3390/brainsci10050274.

[7] G. Corrado and Y. Matias. Multimodal medical artificial intelligence. Posted by Greg Corrado, Head of Health AI, Google Research, and Yossi Matias, VP, Engineering and Research, Google Research, August 2023.

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

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: pre-training of deep bidirectional transformers for language understanding, 2019. arXiv: 1810.04805 [cs.CL].

[10] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. H. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. R. Florence. Palm-e: an embodied multimodal language model. In *International Conference on Machine Learning*, 2023. url: https://api.semanticscholar.org/CorpusID:257364842.

[11] L. Erlani, T. Narawati, Z. Alimin, and S. Tukimin. Using AAC media to improve dynamics ability in music learning for autistic children. *International Journal of Pedagogy and Teacher Education (IJPTE)*, 2:103, Focus Issue-July, 2018.

[12] T. Fel, L. Hervier, D. Vigouroux, A. Poche, J. Plakoo, R. Cadene, M. Chalvidal, J. Colin, T. Boissin, L. Bethune, A. Picard, C. Nicodeme, L. Gardes, G. Flandin, and T. Serre. Xplique: a deep learning explainability toolbox. *Workshop on Explainable Artificial Intelligence for Computer Vision (CVPR)*, 2022.

[13] T.-J. Fu, L. Li, Z. Gan, K. Lin, W. Y. Wang, L. Wang, and Z. Liu. Violet : end-to-end video-language transformers with masked visual-token modeling, 2022. arXiv: 2111.12681 [cs.CV].

[14] L. F. Garcia, L. C. de Oliveira, and D. M. de Matos. Evaluating pictogram prediction in a location-aware augmentative and alternative communication system. *Assistive Technology*, 28(2):83–92, 2016. doi: 10.1080/10400435.2015.1092181. url: https://doi.org/10.1080/10400435.2015.1092181. PMID: 26479456.

[15] K. Han, Y. Wang, J. Guo, Y. Tang, and E. Wu. Vision gnn: an image is worth graph of nodes, 2022. arXiv: 2206.00272 [cs.CV].

[16] R. Hervás, S. S. Bautista, G. Méndez, P. Galvan, and P. Gervás. Predictive composition of pictogram messages for users with autism. *Journal of Ambient Intelligence and Humanized Computing*, 11:5649–5664, 2020. url: https://api.semanticscholar.org/CorpusID:218809350.

[17] Z. Hu, A. Iscen, C. Sun, Z. Wang, K.-W. Chang, Y. Sun, C. Schmid, D. A. Ross, and A. Fathi. Reveal: retrieval-augmented visual-language pre-training with multi-source multimodal knowledge memory. *CVPR*, 2023.

[18] S. Huang, L. Dong, W. Wang, Y. Hao, S. Singhal, S. Ma, T. Lv, L. Cui, O. K. Mohammed, B. Patra, Q. Liu, K. Aggarwal, Z. Chi, J. Bjorck, V. Chaudhary, S. Som, X. Song, and F. Wei. Language is not all you need: aligning perception with language models, 2023. arXiv: 2302.14045 [cs.CL].

[19] A. Ji, N. Kojima, N. Rush, A. Suhr, W. K. Vong, R. D. Hawkins, and Y. Artzi. Abstract visual reasoning with tangram shapes, 2022. arXiv: 2211.16492 [cs.CL].

[20] C. Johnson, S. Myers, and American Academy of Pediatrics Council on Children With Disabilities. Identification and evaluation of children

with autism spectrum disorders. *Pediatrics*, 120(5):1183–1215, 2007. doi: 10.1542/peds.2007-2361.

[21] W. Kim, B. Son, and I. Kim. Vilt: vision-and-language transformer without convolution or region supervision, 2021. arXiv: 2102.03334 [stat.ML].

[22] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, M. S. Bernstein, and F.-F. Li. Visual genome: connecting language and vision using crowdsourced dense image annotations, 2016. arXiv: 1602.07332 [cs.CV].

[23] H. Laurençon, L. Saulnier, L. Tronchon, S. Bekman, A. Singh, A. Lozhkov, T. Wang, S. Karamcheti, A. M. Rush, D. Kiela, M. Cord, and V. Sanh. Obelics: an open web-scale filtered dataset of interleaved image-text documents, 2023. arXiv: 2306.16527 [cs.IR].

[24] W. Lei, Y. Ge, J. Zhang, D. Sun, K. Yi, Y. Shan, and M. Z. Shou. Vitlens: towards omni-modal representations, 2023. arXiv: 2308.10185 [cs.CV].

[25] A. Lekkas, P. Schneider-Kamp, and I. Augenstein. Multi-sense language modelling. *CoRR*, abs/2012.05776, 2020. arXiv: 2012.05776. url: https://arxiv.org/abs/2012.05776.

[26] Y. Levine, B. Lenz, O. Dagan, O. Ram, D. Padnos, O. Sharir, S. Shalev-Shwartz, A. Shashua, and Y. Shoham. SenseBERT: driving some sense into BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4656–4667, Online. Association for Computational Linguistics, July 2020. doi: 10.18653/v1/2020.acl-main.423. url: https://aclanthology.org/2020.acl-main.423.

[27] Lightning AI. Pytorch memory efficient vision-transformer large language model tutorial. https://lightning.ai/pages/community/tutorial/pytorch-memory-vit-llm/, 2023.

[28] B. Y. Lin, W. Zhou, M. Shen, P. Zhou, C. Bhagavatula, Y. Choi, and X. Ren. Commongen: a constrained text generation challenge for generative commonsense reasoning, 2020. arXiv: 1911.03705 [cs.CL].

[29] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: common objects in context, 2015. arXiv: 1405.0312 [cs.CV].

[30] B. MacWhinney. *The Childes Project: Tools for Analyzing Talk, Volume II: the Database*. Psychology Press, 3rd edition, 2000. doi: 10.4324/9781315805641. url: https://doi.org/10.4324/9781315805641.

[31] F. Martínez-Santiago, M. Á. García-Cumbreras, A. Montejo-Ráez, and M. C. Díaz-Galiano. Pictogrammar: an AAC device based on a semantic grammar. In *Proceedings of the 11th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 142–150, San Diego, CA. Association for Computational Linguistics, June 2016. doi: 10.18653/v1/W16-0516. url: https://aclanthology.org/W16-0516.

[32] J. H. Moon, H. Lee, W. Shin, Y.-H. Kim, and E. Choi. Multi-modal understanding and generation for medical images and text via vision-language pre-training. *IEEE Journal of Biomedical and Health Informatics*, 26(12):6070–6080, December 2022. doi: 10.1109/jbhi.2022.3207502. url: https://doi.org/10.1109%2Fjbhi.2022.3207502.

[33] G. Moro, S. Salvatori, and G. Frisoni. Efficient text-image semantic search: a multi-modal vision-language approach for fashion retrieval. *Neurocomputing*, 538:126196, 2023. issn: 0925-2312. doi: https://doi.org/10.1016/j.neucom.2023.03.057. url: https://www.sciencedirect.com/science/article/pii/S092523122300303X.

[34] L. Mottron and D. Gagnon. Prototypical autism: new diagnostic criteria and asymmetrical bifurcation model. *Acta Psychologica*, 237:103938,

2023. issn: 0001-6918. doi: https://doi.org/10.1016/j.actpsy.2023.103938. url: https://www.sciencedirect.com/science/article/pii/S0001691823001142.

[35] D. A. MS, OTR, P. M. OTR, M. P. OTR, H. W. OTR, H. S. OTR, and H. M. OTR. The effects of word completion and word prediction on typing rates using on-screen keyboards. *Assistive Technology*, 18(2):146–154, 2006. doi: 10.1080/10400435.2006.10131913. url: https://doi.org/10.1080/10400435.2006.10131913. PMID: 17236473.

[36] OpenAI. Gpt-4 technical report, 2023. arXiv: 2303.08774 [cs.CL].

[37] V. Ordonez, G. Kulkarni, and T. Berg. Im2text: describing images using 1 million captioned photographs. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. url: https://proceedings.neurips.cc/paper/2011/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

[38] S. Papandrea, A. Raganato, and C. Delli Bovi. SupWSD: a flexible toolkit for supervised word sense disambiguation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 103–108, Copenhagen, Denmark. Association for Computational Linguistics, September 2017. doi: 10.18653/v1/D17-2018. url: https://aclanthology.org/D17-2018.

[39] J. Pereira, N. Franco, and R. Fidalgo. A semantic grammar for augmentative and alternative communication systems. In P. Sojka, I. Kopeček, K. Pala, and A. Horák, editors, *Text, Speech, and Dialogue*, pages 257–264, Cham. Springer International Publishing, 2020.

[40] J. A. Pereira, D. Macêdo, C. Zanchettin, A. L. I. de Oliveira, and R. do Nascimento Fidalgo. Pictobert: transformers for next pictogram prediction. *Expert Systems with Applications*, 202:117231, 2022. issn: 0957-4174. doi: https://doi.org/10.1016/j.eswa.2022.117231. url: https://www.sciencedirect.com/science/article/pii/S095741742200611X.

[41] C. E. Saturno, A. R. G. Ramirez, M. J. Conte, M. Farhat, and E. C. Piucco. An augmentative and alternative communication tool for children and adolescents with cerebral palsy. *Behaviour & Information Technology*, 34(6):632–645, 2015. doi: 10.1080/0144929X.2015.1019567. url: https://doi.org/10.1080/0144929X.2015.1019567.

[42] D. Schwab, P. Trial, C. Vaschalde, L. Vial, E. Esperança-Rodier, and B. Lecouteux. Providing semantic knowledge to a set of pictograms for people with disabilities: a set of links between WordNet and Arasaac: Arasaac-WN. In *LREC*, Marseille, France, 2020. url: https://hal.science/hal-02888279.

[43] S. Schwettmann, N. Chowdhury, and A. Torralba. Multimodal neurons in pretrained text-only transformers, 2023. arXiv: 2308.01544 [cs.CV].

[44] 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)*, pages 2556–2565, Melbourne, Australia. Association for Computational Linguistics, July 2018. doi: 10.18653/v1/P18-1238. url: https://aclanthology.org/P18-1238.

[45] M. Shukor, C. Dancette, A. Rame, and M. Cord. Unified model for image, video, audio and language tasks. *arXiv preprint arXiv:2307.16184*, 2023.

[46] N. Studer, R. Gundelfinger, T. Schenker, and H.-C. Steinhausen. Implementation of early intensive behavioural intervention for children with autism in switzerland. *BMC Psychiatry*, 17(1):34, January 21, 2017. issn: 1471-244X. doi: 10.1186/s12888-017-1195-4. url: https://doi.org/10.1186/s12888-017-1195-4.

[47] Q. Sun, Q. Yu, Y. Cui, F. Zhang, X. Zhang, Y. Wang, H. Gao, J. Liu, T. Huang, and X. Wang. Generative pretraining in multimodality, 2023. arXiv: 2307.05222 [cs.CV].

[48] R. T and P. Sujatha. A survey: approaches for detecting the autism spectrum disorder. *International Journal of Data Informatics and Intelligent Computing*, 2:39–46, June 2023. doi: 10.59461/ijdiic.v2i2.60.

[49] Z. Tang, J. Cho, Y. Nie, and M. Bansal. Tvlt: textless vision-language transformer, 2022. arXiv: 2209.14156 [cs.CV].

[50] O. Thawakar, A. M. Shaker, S. S. Mullappilly, H. Cholakkal, R. M. Anwer, S. S. Khan, J. Laaksonen, and F. S. Khan. Xraygpt: chest radiographs summarization using medical vision-language models. *ArXiv*, abs/2306.07971, 2023. url: https://api.semanticscholar.org/CorpusID:259145194.

[51] D. G. Tosun, İ. Okatan, and H. Köse. Examining the augmentative and alternative communication systems preferences of individuals with autism spectrum disorder. *HAYEF: Journal of Education*, 19(2):146–154, 2022.

[52] C. Wang, Z. Xiao, B. Wang, and J. Wu. Identification of autism based on svm-rfe and stacked sparse auto-encoder. *IEEE Access*, 7:118030–118036, 2019. doi: 10.1109/ACCESS.2019.2936639.

[53] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Ł. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's

neural machine translation system: bridging the gap between human and machine translation, 2016. arXiv: 1609.08144 [cs.CL].

[54] T. Yalim and S. Mohamed. Meltdown in autism: challenges and support needed for parents of children with autism. *International Journal of Academic Research in Progressive Education and Development*, 12(1):850–876, 2023.

[55] S. Yin, C. Fu, S. Zhao, K. Li, X. Sun, T. Xu, and E. Chen. A survey on multimodal large language models, 2023. arXiv: 2306.13549 [cs.CV].

[56] R. Zhang, J. Han, C. Liu, P. Gao, A. Zhou, X. Hu, S. Yan, P. Lu, H. Li, and Y. Qiao. Llama-adapter: efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023.

[57] Z. Zhang, A. Zhang, M. Li, H. Zhao, G. Karypis, and A. Smola. Multimodal chain-of-thought reasoning in language models, 2023. arXiv: 2302.00923 [cs.CL].

[58] H. Zhao, P. Karlsson, D. Chiu, C. Sun, O. Kavehei, and A. McEwan. Wearable augmentative and alternative communication (waac): a novel solution for people with complex communication needs. 27(3), 2023. doi: 10.1007/s10055-023-00818-8. url: https://doi.org/10.1007/s10055-023-00818-8.

[59] C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He, H. Peng, J. Li, J. Wu, Z. Liu, P. Xie, C. Xiong, J. Pei, P. S. Yu, and L. Sun. A comprehensive survey on pretrained foundation models: a history from bert to chatgpt, 2023. arXiv: 2302.09419 [cs.AI].

Acknowledgements

To Prof. Moro and Dr. Frisoni, I extend my heartfelt appreciation for affording me this invaluable opportunity and guiding me through every step of its realization. I'm grateful for the guidance I received from them as the course came to a close.
I would also like to express my gratitude to Dr. Borghi for his unwavering dedication and keen interest in the crucial field of children's learning through AAC systems.

I'm writing to express my appreciation to everyone who has supported me along the way. Your presence has warmed my heart throughout this journey. I'd like to thank my family especially for constantly being beside me.