A TWO-STAGE MACHINE LEARNING SYSTEM FOR THE ANNOTATION OF VISUAL SCHEMAS: MODELS FOR BOUNDARY DETECTION AND MULTI-CLASS CLASSIFICATION OF VISUAL SCHEMAS

Science Fund of the Republic of Serbia
Programme: IDEAS
Project: Structuring Concept Generation with the Help of Metaphor, Analogy, and Schematicity (Project No. 7715934)
Acronym: *SCHEMAS*
Prepared by: Mladen Popović, MSc

1. Introduction

Over the past several decades, cognitive linguistics has highlighted the complex interrelationship between conceptual structures and linguistic expressions, demonstrating that language is not an autonomous formal system but is instead deeply rooted in embodied cognition (Evans & Green, 2006; Langacker, 2008). Since the early work of Lakoff, Johnson, and their contemporaries, it has become a foundational principle that meaning is grounded in sensorimotor experience and shaped by patterns of bodily interaction with the environment (Johnson, 1987; Lakoff, 1987). Within this theoretical framework, image schemas have emerged as a particularly important concept: they represent recurring patterns of perception and movement that influence the structuring of more complex conceptualizations (Lakoff, 1987: 28–29). Image schemas are conceptualized as basic, pre-linguistic gestalts arising from embodied experience and recurring across multiple conceptual domains, providing a framework upon which more complex metaphorical and abstract reasoning is constructed (Lakoff, 1987: 29–30).

Although their role in conceptual meaning and metaphorical extension has been extensively studied, comparatively less attention has been devoted to how contemporary computational approaches can model the occurrence of image schemas in texts. This gap not only raises theoretical questions but also poses a practical challenge for computational approaches to language: how can we design models that automatically detect these image-schematic patterns in natural texts? Addressing such questions opens up a fertile field of inquiry. If image schemas are indeed foundational to our conceptual apparatus, then examining how machine-

learning models learn to associate linguistic sequences with particular schemas may provide deeper insight into how language is constructed, processed, and understood.

At the core of many studies in cognitive linguistics lies the notion of image schemas: dynamic, embodied patterns of experience that arise through repeated sensorimotor interactions with the environment (Johnson, 1987; Lakoff, 1987; Mandler, 2004). Image schemas are not static mental images; rather, they constitute continuous and interactive patterns of experience, such as movement along a path, the application of force, or being located within a bounded space (Johnson, 1987; Lakoff, 1987; Mandler, 2004). According to Johnson (1987), these schemas emerge early in cognitive development, as infants learn to maintain bodily balance, manipulate objects, notice objects entering and exiting containers, and orient themselves in space.

Because image schemas are rooted in bodily experience, they are commonly regarded as universal or near-universal cognitive structures. They arise from embodied interactions shared by all humans, such as the sense of vertical orientation (UP–DOWN) or the experience of boundaries and enclosed spaces (CONTAINMENT) (Johnson, 1987; Evans & Green, 2006). However, the specific linguistic and cultural elaborations of these schemas may vary, allowing a certain degree of universality at the conceptual level to coexist with linguistic diversity.

Despite this rich body of research, scholarly attention to image schemas has largely focused on their conceptual and semantic dimensions. Cognitive linguists have examined in detail the ways in which image schemas shape meaning, conceptual metaphors, and reasoning (Lakoff, 1987; Johnson, 1987), but the question of how these conceptual patterns manifest at the linguistic level − how they are encoded, cued, or reinforced by linguistic structures − has remained relatively underexplored.

Computational linguistics has, in most cases, approached the problem of semantics in natural language from the perspective of word sense disambiguation (using vector-based or Bayesian methods), the establishment of textual entailment, as well as tasks such as segmentation, summarization, and named entity recognition (Dahlgreen, 1998; Helbig, 2005; Kapetanios et al., 2013). Although some early studies pointed to possible correlations between syntactic constructions and fundamental image schemas (Clausner & Croft, 1999; Wachowiak & Gromann, 2022), the field as a whole has not yet fully confronted the challenge of mapping image schemas onto patterns of natural language.

Moreover, the machine-learning perspective on this problem – the development of algorithms for the detection and annotation of potential image-schematic structures in text − remains largely unexplored. Methods proposed in previous research range from hand-crafted, corpus-based algorithms to unsupervised machine-learning approaches that use part-of-speech (POS) tags to identify combinations of nouns, verbs, and prepositions indicative of image schemas (Dodge & Lakoff, 2005; Bennett & Cialone, 2014; Gromann & Hedblom, 2017; Wachowiak, 2020; Wachowiak & Gromann, 2022). The most recent of these approaches employs a variant of BERT (Bidirectional Encoder Representations from Transformers), a language model that represents texts as sequences of vectors using a transformer architecture and attention

mechanisms (including self-attention), which has been further fine-tuned to classify image schemas using entire text sequences as input (Vaswani et al., 2017; Devlin et al., 2019; Wachowiak & Gromann, 2022). However, it should be noted that none of the approaches described here attempts to model the presence of multiple schemas within a given linguistic sequence, nor do they decompose larger structures, such as sentences, into smaller schema-bearing segments.

The central research question of the present study concerns how image schemas, as conceptual building blocks, can be made machine-readable and learnable, such that a computational model can process and identify schemas occurring in any linguistic segment, regardless of its length. If image schemas indeed constitute the foundation of a wide range of conceptual and linguistic phenomena, it is reasonable to assume that their influence can be detected not only at the semantic or conceptual level, but also at the level of individual words and their combinations.

Wachowiak and Gromann (2022), in their discussion of possible models for the annotation of image schemas, emphasize that an adequate image-schema classifier should interact directly with individual words (or their combinations), such that the model's output simultaneously *locates* and *annotates* schematic complexes.

It is also worth noting that, in their discussion of multi-label annotation and its application to image schemas, these authors identify a particular challenge − specifically, the problem of *input scope*. By way of illustration, consider the input to their classifier, which takes an entire sentence as input and returns only a single label. If we assume that multiple image schemas can occur within a single sentence, the classifier would first need to segment the sentence into smaller units and then predict schemas with respect to those individual segments, especially in cases where a schema is realized across multiple words.

To see why this is not a trivial problem, we may consider an example from Wachowiak and Gromann's model (2022), in which the words *beyond*, *attractive*, *answer*, and *white* are labeled as indicators of the CONTAINMENT schema. This appears to be a side effect of the model's failure to clearly distinguish between lexical elements that contribute to the realization of a schema and those that do not. Addressing this issue requires overcoming yet another obstacle: the relative lack of empirical research employing large-scale corpus-based methods to investigate how image schemas are realized in natural language.

Most early work in cognitive linguistics on image schemas relies on introspective data, small sets of examples, or carefully constructed experimental materials (Gibbs et al., 1994; Gibbs & Colston, 1995; Dodge & Lakoff, 2005; Bennett & Cialone, 2014). Consequently, there are no publicly available corpora specifically designed for training schema-classification models, which forces researchers to create suitable datasets *ad hoc*. As a result, training data are limited in size and are most often computationally derived from texts (e.g. Gromann & Hedblom, 2017; Wachowiak & Gromann, 2022). If a hypothetical image-schema classifier is to perform at a satisfactory level, it will require both large-scale data and an efficient processing logic capable of highlighting the relevant elements within a sentence.

To address these questions, the present study draws on the SCHEMAS corpus – a purpose-built dataset compiled for the investigation of image schemas and their linguistic realizations. The SCHEMAS corpus was created through the manual annotation of image schemas in texts drawn from various online newspapers. It consists of two smaller subcorpora, one in Serbian and one in English, and was originally used to examine the distribution of image schemas across different languages. In the present context, this carefully prepared corpus enables us to train (1) a processing model that identifies meaning-bearing segments of text (i.e. textual "chunks") in which schemas occur, as well as (2) a multi-label classification model trained on naturalistic data. The decision to use an already annotated corpus is primarily motivated by the fact that it allows us to avoid issues inherent to synthetic data, while at the same time providing sufficiently diverse and realistic input that reduces the risk of overfitting.

Crucially, both models must be context-sensitive in order to handle the simultaneous occurrence of multiple schemas effectively. Consider, for example, the sequence "*He was falling into a deep depression.*" An annotator following the SCHEMAS project guidelines might annotate the entire sequence with schema labels, placing the tags <PATH><FORCE><CONTAINMENT> after the word "*depression.*" It is worth noting that the primary contributions to the <PATH> and <CONTAINMENT> schemas are found in the segment "*was falling ... into a deep depression*," whereas the main contribution to the <FORCE> schema is localized in the word "*falling.*"

Thus, for a model to accurately predict the occurrence of these schemas, it must be capable of contextual understanding. To capture context, both the segmentation model and the classifier model were built using BERT, a deep neural language model based on the now widely adopted transformer architecture (Vaswani et al., 2017; Devlin et al., 2019). BERT processes context using the transformer's self-attention mechanism in a bidirectional manner. This means that, unlike models such as GPT, BERT processes tokens in both directions simultaneously, so that word representations during training are learned by taking into account tokens to both the left and the right of the target word.

This is achieved through the transformer's multi-head self-attention mechanism, which allows each token to attend to every other token by computing attention weights that indicate the relevance of each token with respect to the token being processed. The architecture is integrated during the training process, in which BERT randomly masks input tokens with the goal of later predicting the masked words. Because prediction is performed by taking into account tokens both to the left and to the right of the target words, the model is able to learn deep contextual interactions from all directions. After training is completed, the representation of each token also encodes information about all the other tokens in the input sequence. These representations (embeddings) are subsequently fine-tuned for specific tasks such as identifying schema-bearing segments and annotating schemas, as is the case in the present study. This is accomplished using a feed-forward classifier (often referred to as a dense or linear layer), a small neural network module placed on top

of BERT's final output, which transforms its high-dimensional contextual encodings into task-specific predictions (Devlin et al., 2019).

Particularly important for our task is the fact that BERT, as a result of its training objective involving masked word prediction within a sentence, acquires a rich, contextualized representation of language. Moreover, once the model has been pretrained, it can be further adapted to a specific task by adding an additional layer. Finally, the model is optimized in an end-to-end manner, meaning that no additional feature engineering is required; instead, the model autonomously learns to attend to the features most relevant for the given task. Given that our corpus contains annotations that correspond to more or less clearly defined text sequences, the attention mechanism is considered a particularly important component of the overall system (Wachowiak & Gromann, 2022).

In the remainder of the paper, this study will first attempt to model the procedure followed by a human annotator, in order to highlight some of the requirements that the computational model should satisfy. It will then describe the process of preparing the corpus for model training and testing, as well as the overall structure of the solution (the processing pipeline). The final part of the study will demonstrate how the two models operate in synergy and will propose possible improvements. One of the main drivers of the overall model architecture was the set of observations presented in Wachowiak and Gromann (2022); accordingly, we will occasionally refer back to their solution in what follows, as it represented the closest available analogue to our own at the time of writing.

2. The method

Let us now consider the task of the human annotator. Upon encountering a text, a hypothetical annotator, following the SCHEMAS project annotation guidelines (as outlined in Antović et al., 2023; Figar & Veličković, 2023), might read the text and, taking into account the examples provided in the guidelines, annotate a sample sentence as follows:

> [...] "This legislation is a giant step forward<ms><F><P++><spec><forward> in our fight to combat<ms><F+><L> the fentanyl crisis, crack down on the dealers peddling<ms><F><P+><L> death in our communities, and accelerate<ms><F+><P+><spec><forward><L> our state's public health response to get this deadly drug off our streets<s><F+><P><L--> and save lives," House Speaker Alec Garnett, a Democrat, said after the bill's passage<ms><F><P+><spec><end path>." [...]

Sequences beginning with <ms> or <s> and ending with the final symbol (>) represent examples of schematic complexes. In this case, they indicate that a

particular stretch of text has been recognized as containing schemas − specifically, <FORCE>, <PATH>, <LINK>, <BALANCE>, <CONTAINMENT> − as well as a scalar modification (a <SCALE> schema that "applies" to these five core schemas). Scalar modifications in the annotation set are marked with either a plus or minus sign, indicating whether the valence is positive or negative, with multiple repeated signs denoting greater intensity (for example, a <FORCE> schema with one plus sign is interpreted as more intense than the same schema without a sign).

The presence of multiple annotation clusters (conceptual combinations composed of several image schemas) within a sentence indicates that the annotator has associated specific words from the sentence with particular schematic clusters. For example, the sentence "*This legislation is a giant step forward*" is annotated as <ms><F><P++><spec><forward>, whereas "*in our fight to combat*" is annotated as <ms><F+><L>. When text is annotated in this manner, it becomes possible to conduct further analyses to determine both absolute and relative frequencies of schemas, either as clusters or as individual schemas.

It should be noted, however, that a single sentence can contain multiple clusters, while no cluster in the corpus receives information from elements outside the boundaries of that sentence. It is possible that annotators, while reading the final segments of one sentence, carried over information into the next, resulting in a "bleed-over" of information between structures that, in theory, should constitute separate meaning units. Nevertheless, for the purposes of model training, it was decided that sentence boundaries were to be retained as "hard" constraints, based on a preliminary analysis of segmentation (chunking) logic, which showed that this approach successfully preserves semantically significant segments in all 300 randomly selected examples from the corpus. Accordingly, a potential annotation procedure can be represented approximately as follows:

1. The annotator familiarizes themselves with the protocol and criteria for identifying each image schema.
2. The annotator keeps these criteria "active" in memory while reading and processing the incoming text.
3. During reading, the processing of specific lexical elements and their contextual frames increases the likelihood that certain schemas will be detected and annotated.
4. When this cumulative probability reaches a certain threshold, the schemas are recognized and recorded in the annotation. This accumulation occurs simultaneously for multiple schemas.
5. The moment of annotation interrupts the existing accumulation of probability, meaning that after annotating a particular segment within a sentence, the process restarts from the beginning.
6. Since no schemas extend beyond sentence boundaries, only local sequences within the sentence itself are relevant for accumulation.

Let us consider how a computational variant of the annotator could emulate a human annotator. At the most basic level, the system must process each input text

and segment it into sentences. Each sentence is then further divided into individual tokens, allowing an attention-like mechanism to track interactions between tokens and gradually accumulate probabilities for the presence of specific schemas within a segment. Once these probabilities exceed a certain threshold, the computational annotator would finalize the chunk and assign the relevant schema labels to that segment, in a manner analogous to how a human annotator stops reading, records the recognized schemas, and then continues. With this in mind, the first step in developing our predictive model consists of formalizing a method that reliably "captures" the portion of text located to the left of a given schematic cluster.

To train a BERT-based model, the corpus must be segmented so that each schematic cluster is clearly associated with its direct "source," i.e., the text contributing to its detection. To this end, we used the NLTK library to create an algorithm that searches the text to the left of a cluster and stops when it encounters a sentence boundary or another schematic cluster (Bird, 2006). Sentence segments contributing to a given cluster are referred to as "segments" or "chunks," as they often include extraneous information. These segments are further processed by assigning labels to the tokens: the onset token of each segment receives a specific label, while all the other tokens within the segment are assigned interior token labels. Tokens that do not contribute to schema recognition are labeled as empty tokens. These labels are then fed into a modified version of BERT, based on the "BERT base uncased" model available on the Hugging Face platform (Wolf et al., 2020). The task of identifying these segments is, by its nature, similar to other sequence classification tasks, for which BERT has already demonstrated reliable performance (Wolf et al., 2020).

Because chunks can exhibit significant syntactic variation − including fully formed syntactic structures and extraneous fragments (e.g., "the fentanyl crisis, crack down on the dealers peddling<ms><F><P+><L>") − the computational annotator must also "learn" how to selectively evaluate or ignore portions of each chunk. In other words, the system cannot rely solely on a single deterministic algorithm; rather, it must adapt to different linguistic configurations through dynamic adjustment of the probabilities associated with encountering particular schemas. This adaptive capacity is precisely what makes a machine learning approach indispensable, as it allows the model to generalize from annotated examples and handle the inherent variability of language in natural contexts. Similar to the task of identifying schema-bearing chunks, a BERT model was again chosen as the foundation, trained on textual chunks obtained using the previously described processing algorithm. Each schematic cluster associated with a chunk was also extracted and encoded as a "one-hot" vector (Brownlee, 2020).

All of the code was implemented in Python, and the corresponding Jupyter notebook is available at the following address:

https://github.com/MladenPopovicFilFak/ProjectSchemasAnnotator.

The segmentation model and the annotation model can be found here:

https://huggingface.co/MladenIgnatum/Segmentation_Model https://huggingface.co/MladenIgnatum/Annotation_Model.

3. Text segmentation tool – From corpus to training sets

The first step in constructing our synthetic annotator involves preprocessing the annotated text so that the resulting dataset preserves meaningful structures (i.e., chunks and their corresponding annotations) while remaining machine-readable. At the most basic level, the parsing algorithm for the human-annotated corpus must:

1. Preserve the portion of the text located to the left of the annotation, and
2. Preserve the annotation itself.

This directly informs the architecture of the annotation model. Considering the model requirements to both segment text and perform annotation, it was decided that the process should proceed in two phases. First, a submodel based on BERT is trained on the first data segment (1) to learn to identify chunks containing schemas and extract them from any input text. Then, a second submodel, also based on BERT, is trained on the textual segments from (1), together with their corresponding schematic clusters (2). This second model used the output of the first submodel to predict the presence of schemas. The code and instructions are detailed below.

Cell 1 handles the import of the necessary libraries. **chardet** is used for encoding detection, **nltk** for linguistic parsing, while **torch** and **scikitlearn** are used for managing the learning environment. The **transformers** library is employed for working with BERT and related operations (Chardet n.d.; Collobert et al., 2002; Bird, 2006; Pedregosa et al., 2011; Duchesnay, 2011).

Boundary labels are used to annotate segments carrying schemas. The first token of a segment is labeled the B-Chunk (indicating the beginning of a given segment), I-Chunk labels denote words between the start of the segment and the associated schematic cluster, and the O label applies to tokens that do not contain schemas.

```
################################################
CELL 1: SETUP, INSTALLS, AND IMPORTS
################################################

from google.colab import drive
drive.mount('/content/drive')

import os
import re
```

```
import chardet
import nltk
import torch
import numpy as np
from nltk import sent_tokenize
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score

from transformers import (
  AutoTokenizer,
  AutoModelForTokenClassification,
  AutoModelForSequenceClassification,
  Trainer,
  TrainingArguments
)

# Download NLTK sentence tokenizer data
nltk.download('punkt')
nltk.download('punkt_tab')

# Global configs
MODEL_NAME = "bert-base-uncased"   # Or another HF model
MAX_LEN = 128

# For Stage 1 boundary detection
BOUNDARY_LABELS = ["O", "B-CHUNK", "I-CHUNK"]
label2id_boundary = {lab: i for i, lab in enumerate(BOUNDARY_LABELS)}
id2label_boundary = {i: lab for i, lab in enumerate(BOUNDARY_LABELS)}

# For Stage 2 multi-label classification
SCHEMA_LABELS = ["P", "F", "L", "B", "C"]
SCHEMA2ID = {s: i for i, s in enumerate(SCHEMA_LABELS)}
```

Cell 2 is responsible for parsing the raw textual data to identify and extract meaningful segments based on the inline annotations. This parsing logic is crucial for preparing the data for both phases of the procedure − **phase 1** (boundary detection) and **phase 2** (schema classification). Specifically, **cell 2**:

- **Splits the text into segments:** Uses regular expressions to identify and separate parts of the text marked with inline annotations (e.g., <P>, <F+>).
- **Processes the annotations:** Simplifies complex labels into basic schema labels, ensuring consistency and relevance.
- **Prepares data for training:** Structures the data into formats suitable for boundary detection and schema classification tasks.

```
#####################################################
CELL 2: PARSING LOGIC FOR INLINE TAGS -> CHUNKS
#####################################################

# Define SCHEMA_LABELS and SCHEMA2ID (Ensure consistency)
SCHEMA_LABELS = ["P", "F", "L", "B", "C"]
SCHEMA2ID = {s: i for i, s in enumerate(SCHEMA_LABELS)}
SCHEMA_ID2LABEL = {i: s for i, s in enumerate(SCHEMA_LABELS)}

def parse_clusters_in_sentence(sentence):
  """
  Splits a sentence on consecutive <...> tags.
  Associates each tag group with the preceding chunk.
  Returns a list of dicts:
   [
    {
     "chunk_text": "...",
     "raw_tags": [...],
    },
    ...
   ]
  """
  # Pattern to identify consecutive tags as one group
  pattern = r'((?:<[^>]+>)+)'
  parts = re.split(pattern, sentence)

  chunks = []

  # Iterate over parts in pairs: text + tags
  for i in range(0, len(parts), 2):
    text_chunk = parts[i].strip()
    tags = []
```

```
        if i + 1 < len(parts):
            tags = re.findall(r'<([^>]+)>', parts[i + 1])
        if text_chunk:
            chunks.append({
                'chunk_text': text_chunk,
                'raw_tags': tags.copy()
            })

    return chunks

def collapse_raw_tags(raw_tags):
    """
    Convert tags like ["ms", "F+", "P++"] to a set of [P, F, L, B, C].
    Discard irrelevant tags (ms, spec, forward, etc.).
    """
    core_set = set()
    for rt in raw_tags:
        if not rt:
            continue
        base = rt[0].upper()  # Ensure case insensitivity
        if base in {"P", "F", "L", "B", "C"}:
            core_set.add(base)
    return core_set

def parse_text_into_stage_data(raw_text, tokenizer):
    """
    1) Sentence-splits the text.
    2) For each sentence, parse chunk boundaries (Stage 2) + create
B/I/O (Stage 1).
    Returns:
      stage1_data: [{"tokens": [...], "labels": ["B-CHUNK","I-CHUNK",...]}]
      stage2_data: [{"chunk": "...", "labels": [0/1,...]}]
    """
    sentences = sent_tokenize(raw_text)
    stage1_data = []
    stage2_data = []

    for sent_idx, sent in enumerate(sentences, 1):
        # Identify chunk boundaries
        sent_chunks = parse_clusters_in_sentence(sent)
        print(f"\nProcessing Sentence {sent_idx}: {sent}")
```

```
    print(f"Identified Chunks: {sent_chunks}")

    # Reconstruct clean_sentence without tags
    clean_sentence = re.sub(r'<[^>]+>', '', sent).strip()
    print(f"Clean Sentence: {clean_sentence}")

    # Tokenize the clean_sentence with offset mapping
    encoding = tokenizer(clean_sentence,
return_offsets_mapping=True, add_special_tokens=False)
    tokens = tokenizer.convert_ids_to_tokens(encoding['input_ids'])
    offset_mappings = encoding['offset_mapping']
    print(f"Tokens: {tokens}")
    print(f"Offset Mappings: {offset_mappings}")

    # Initialize labels as "O"
    label_sequence = ["O"] * len(tokens)

    # Track the last assigned character to prevent overlapping
assignments
    last_assigned_char = 0

    for ch_idx, ch in enumerate(sent_chunks, 1):
      chunk_text = ch["chunk_text"]
      raw_tag_set = collapse_raw_tags(ch["raw_tags"])

      if not raw_tag_set:
        # This chunk has no labels, so tokens remain "O"
        print(f"Chunk {ch_idx}: '{chunk_text}' - No Labels Assigned")
        continue  # Do not assign labels to these tokens

      # Find the chunk_text in clean_sentence starting from
last_assigned_char
      start_char = clean_sentence.find(chunk_text, last_assigned_char)
      if start_char == -1:
        print(f"Warning: Chunk '{chunk_text}' not found in
clean_sentence.")
        continue

      end_char = start_char + len(chunk_text)
      print(f"Chunk {ch_idx}: '{chunk_text}' - Start: {start_char}, End:
{end_char}")
```

```
        # Assign labels to tokens within [start_char, end_char)
        first_token = True
        for i, (token_start, token_end) in enumerate(offset_mappings):
            if token_start >= start_char and token_end <= end_char:
                if label_sequence[i] == "O":  # Only assign if not already
labeled
                    if first_token:
                        label_sequence[i] = "B-CHUNK"
                        first_token = False
                    else:
                        label_sequence[i] = "I-CHUNK"

        # Assign Stage 2 labels
        label_vec = [0] * len(SCHEMA_LABELS)
        for t in raw_tag_set:
            if t in SCHEMA2ID:
                idx = SCHEMA2ID[t]
                label_vec[idx] = 1
        print(f"Raw Tags: {ch['raw_tags']}")
        print(f"Assigned Labels: {label_vec}")

        stage2_data.append({
            "chunk": chunk_text,
            "labels": label_vec
        })

        # Update last_assigned_char to end_char to prevent overlapping
        last_assigned_char = end_char

    # Append to Stage1 data
    stage1_data.append({
        "tokens": tokens,
        "labels": label_sequence
    })

  return stage1_data, stage2_data
```

In **cell 2**, the following elements are present: **SCHEMA2ID** and **SCHEMA_ID2LABEL**, dictionaries that map schema labels to unique IDs and vice versa. These mappings are crucial for model training and prediction, as they provide a consistent numerical representation of the labels (Collins & Syme, 1995).

```
# Define SCHEMA_LABELS and SCHEMA2ID (Ensure consistency)
SCHEMA_LABELS = ["P", "F", "L", "B", "C"]
SCHEMA2ID = {s: i for i, s in enumerate(SCHEMA_LABELS)}
SCHEMA_ID2LABEL = {i: s for i, s in enumerate(SCHEMA_LABELS)}
```

The function parse_clusters_in_sentence(sentence) uses regular expressions to detect and group consecutive inline tags within a sentence. The purpose of this function is to identify and group consecutive inline tags in a single sentence and link each group of tags to the preceding textual segment, effectively creating a mapping of text chunks to their corresponding labels. The regular expression pattern r'((?:<[^>]+>)+)' works as follows:

1. Defines a non-capturing group (?:) to exclude the text to the left of the first annotation,
2. Recognizes any sequence of characters within < and > as a single tag <[^>]+>,
3. Matches one or more consecutive tags (?:<[^>]+>)+,
4. Captures the entire sequence of consecutive tags ((?:<[^>]+>)+).

After that, re.split splits the sentence into alternating segments of text and groups of tags. For example, "This is a sample <P>text<F+>." would be split into: ['This is a sample ', '<P><F+>', '.'] (Van Rossum, 2020). The function then iterates through these segments and creates a dictionary containing the text chunks and their raw tags. For example: [{ "chunk_text": "This is a sample", "raw_tags": ["P", "F+"] }]

The function collapse_raw_tags(raw_tags) simplifies the schemas present in the corpus. Considering that the number of possible schemas and their combinations is quite large, the pipeline was initially tested using a reduced set of schemas. All specific tags, such as <forward>, <up>, <down>, etc., were removed, while scalar schemas were consolidated into their base variant: <F+>, <F++>, <F+++>, and other scalar versions were collapsed into a single category <F>.

```
def parse_clusters_in_sentence(sentence):
    """
    Splits a sentence on consecutive <...> tags.
    Associates each tag group with the preceding chunk.
    Returns a list of dicts:
     [
      {
       "chunk_text": "...",
       "raw_tags": [...],
      },
      ...
     ]
    """
    # Pattern to identify consecutive tags as one group
    pattern = r'((?:<[^>]+>)+)'
    parts = re.split(pattern, sentence)

    chunks = []

    # Iterate over parts in pairs: text + tags
    for i in range(0, len(parts), 2):
        text_chunk = parts[i].strip()
        tags = []
        if i + 1 < len(parts):
            tags = re.findall(r'<([^>]+)>', parts[i + 1])
        if text_chunk:
            chunks.append({
                'chunk_text': text_chunk,
                'raw_tags': tags.copy()
            })

    return chunks

def collapse_raw_tags(raw_tags):
    """
    Convert tags like ["ms", "F+", "P++"] to a set of [P, F, L, B, C].
    Discard irrelevant tags (ms, spec, forward, etc.).
    """
    core_set = set()
    for rt in raw_tags:
        if not rt:
```

```
        continue
    base = rt[0].upper()  # Ensure case insensitivity
    if base in {"P", "F", "L", "B", "C"}:
        core_set.add(base)
return core_set
```

The function parse_text_into_stage_data(raw_text, tokenizer) processes raw input texts and creates datasets for training both models. The function outputs two lists of dictionaries: **stage1_data** contains tokens along with their corresponding **boundary labels**, which indicate the start, inside, or outside of a chunk, while **stage2_data** contains the extracted text chunks and their corresponding schemas, represented as multi-hot vectors that indicate the presence or absence of a particular schema (Collins & Syme, 1995). To illustrate, the function returns data of the following form:

```
stage1_data = [
  {
    "tokens": ["Brent", "crude", "'", "s", "rise", "above",
"that", "milestone", "."],
    "labels": ["B-CHUNK", "I-CHUNK", "O", "O", "O", "O",
"O", "O", "O"]
  },
  ...
]

stage2_data = [
  {
    "chunk": "Brent crude's rise above that milestone",
    "labels": [1, 1, 0, 0, 0]  # Example: P and F schemas
present
  },
  ...
]
```

So, if the sentence: “Today another American president faces rising<ms><P><F><Spec><UP> fuel prices, spurred<ms><F+><P++><L+> by a challenge mostly out of his control, an invasion<s><F++><P++><L++><C+> of Ukraine by Russia, a top oil and gas producer intent to use its energy supplies as a weapon when necessary.” is processed by the function, the output of the processing looks like this:

Stage 1 Data: Sentence 1: today: B-CHUNK another: I-CHUNK american: I-CHUNK president: I-CHUNK faces: I-CHUNK rising: I-CHUNK fuel: B-CHUNK prices: I-CHUNK ,: I-CHUNK spurred: I-CHUNK by: B-CHUNK a: I-CHUNK challenge: I-CHUNK mostly: I-CHUNK out: I-CHUNK of: I-CHUNK his: I-CHUNK control: I-CHUNK ,: I-CHUNK an: I-CHUNK invasion: I-CHUNK of: O ukraine: O by: O russia: O ,: O a: O top: O oil: O and: O gas: O producer: O intent: O to: O use: O its: O energy: O supplies: O as: O a: O weapon: O when: O necessary: O .: O

and:

Stage 2 Data:
Chunk 1: Today another American president faces rising
Assigned Labels: ['P', 'F']

Chunk 2: fuel prices, spurred
Assigned Labels: ['P', 'F', 'L']

Chunk 3: by a challenge mostly out of his control, an invasion
Assigned Labels: ['P', 'F', 'L', 'C']

The data from **phase 1** are used for training and validating the segmentation model, while data from **phase 2** are used for training and validating the annotation model. The models are applied in tandem, so that each input text is first segmented using the segmentation model, and then the resulting chunks are passed to the schema-assignment model. This approach allows efficient and precise handling of textual data in two stages, with the goal of ensuring the best possible interpretation and classification of complex schematic structures in the text.

Cell 3 is responsible for transforming unstructured textual data into a structured format, thereby laying the foundation for efficient model training in the following cells. The code in this cell applies previously defined functions and iterates through all files that make up our corpus:

- **Reading raw text files:** It goes through the specified directories to locate and load all .txt files containing raw data.
- **Detection and handling of encoding:** Uses the **chardet** library for accurate detection of each text file's encoding, ensuring correct reading of diverse datasets (**chardet, n.d.**).
- **Parsing text into structured data:**
  - **Phase 1 (Boundary Detection):** Prepares token-level labeled data (BIO — Begin, Inside, Outside) for recognizing chunk boundaries.
  - **Phase 2 (Schema Classification):** Structures data with multi-hot labels corresponding to predefined schema categories for each identified chunk (Wachowiak & Gromann, 2022).

Cell 3 completes the data preprocessing step in the overall procedure. The procedure contained in the first three cells can accept any annotated file as input, provided that the annotation procedure is similar to that used in the original files. The structure of the procedure is as follows: (1) Raw texts with annotations are split into textual chunks associated with specific schema groupings, and (2) for the first model, a dataset is created using boundary labels, while for the second model, a dataset is created by encoding schema groupings as one-hot vectors.

Cell 4 further processes these two datasets so that they can be used as input values for the two models. **Cells 5** and **6** initialize and train these two models: **cell 5** handles the segmentation model, while **cell 6** handles the annotation model. Finally, **cell 7** applies these two models in tandem. In **cell 7**, input text is provided for processing, after which the models jointly perform segmentation and annotation of the input text.

```
##################################################
CELL 3: READ ALL TXT FILES, DETECT ENCODING, PARSE -> STAGE1 &
STAGE2
##################################################
# Define your tokenizer (ensure it matches the one used in parsing logic)
MODEL_NAME = "bert-base-uncased"  # Replace with your specific
model if different
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
FOLDER_PATH = "/content/drive/"  # Change depending on the location
of your input files
all_stage1 = []
```

```
all_stage2 = []
txt_files = [f for f in os.listdir(FOLDER_PATH) if f.endswith('.txt')]
print(f"Found {len(txt_files)} text files.")
for filename in txt_files:
    full_path = os.path.join(FOLDER_PATH, filename)

    # 1) Detect encoding
    with open(full_path, 'rb') as f:
        raw_data = f.read(2048)
        detected = chardet.detect(raw_data)
        encoding = detected['encoding']
        if not encoding:
            encoding = 'utf-8'  # Fallback encoding
            print(f"Encoding not detected for {filename}. Using fallback
encoding 'utf-8'.")

    # 2) Read file with detected encoding
    try:
        with open(full_path, 'r', encoding=encoding, errors='replace') as f:
            file_text = f.read()
    except Exception as e:
        print(f"Error reading {filename} with encoding {encoding}: {e}")
        continue  # Skip to the next file in case of an error

    # 3) Parse text -> stage1, stage2
    stage1_data, stage2_data = parse_text_into_stage_data(file_text,
tokenizer)
    all_stage1.extend(stage1_data)
    all_stage2.extend(stage2_data)

    print(f"Processed file: {filename}")

print(f"\nTotal Stage1 examples: {len(all_stage1)}")
print(f"Total Stage2 examples: {len(all_stage2)}")
```

4. Data for the model and training – segmentation and annotation

The data processed by the algorithm described in cells 1–3 were taken from the SCHEMAS corpus. The specifics of the corpus itself (as described earlier) and the annotation procedure are too numerous to detail here, but one particularly important fact is that each annotator contributed 50,000 words, working in pairs. After completing the annotation, each member of a pair reviewed the annotations of the other member. In the next step of the procedure, another pair (unrelated to the first) additionally verified the annotations of the first pair.

This circumstance significantly influenced the choice of training data – to maintain consistency, it was decided that both models would be trained on a sub-corpus of 100,000 words, composed of the annotations from one pair of annotators. The main assumption was that this choice would preserve a more uniform annotation approach (assuming equal or approximately equal sentence contexts), while simultaneously increasing the size of the training dataset. Furthermore, the number of schemas per category was highly unbalanced, with some being very sparsely represented (e.g., the BALANCE schema), which made it necessary to expand beyond the corpus of a single annotator (50,000 words) to include the rarer schemas. These limitations should be taken into account in subsequent work. **Cells 5**, **6**, and **7** are shown in the appendix because they do not introduce new algorithms.

Cell 4 is responsible for converting the parsed data from **cell 3** into structured datasets suitable for training machine learning models, both in **phase 1** (boundary detection) and **phase 2** (schema classification). Specifically, it:

- Defines custom Dataset classes: creates PyTorch Dataset subclasses for each phase (Collobert et al., 2002).
- Encodes input texts: uses tokenizer[1] to convert textual data into token IDs, attention masks, and other necessary inputs for the models.
- Processes labels appropriately: prepares and formats labels according to model requirements (e.g., BIO labels for token classification and multi-hot vectors for multi-label classification).
- Prepares data for training: organizes the data into a format compatible with the Hugging Face Trainer API (Wolf et al., 2020).

For the boundary detection task, a training set of 3,834 samples and a validation set of 427 samples are used. On the other hand, the schema classification task relies on a training set of 2,851 samples and a validation set of 317 samples.

At the core of the boundary detection module is a token-level classification model based on BERT (Devlin et al., 2017). This model is designed to assign BIO (Begin, Inside, Outside) labels to each token within a given text sequence, thereby determining the boundaries of relevant chunks. Mathematically, for an input token sequence $T = \{t_1, t_2, \ldots, t_n\}$, the model computes contextual embeddings $E = \{e_1, e_2, \ldots, e_n\}$ using the transformer. These embeddings are then passed through a linear layer W and a softmax activation function to obtain the probabilities for the BIO labels of each token:

$$P(l_i \mid t_i) = \text{softmax}(We_i),$$

where l_i denotes the label assigned to token t_i.

In parallel, the schema classification component is designed as a multi-label classification model, which uses a similar transformer-based architecture. Unlike the boundary detection model, this classifier operates at the sequence level, taking

[1] A tokenizer is a tool which transforms raw text into smaller segments known as tokens – such as words, subcategories of words or characters, and then converts those tokens into numeric identifiers which the model can understand and process.

entire text chunks as input and simultaneously predicting the presence of multiple schema categories. For a given input chunk $C = \{c_1, c_2, \ldots, c_m\}$, the model generates an encoding h_{CLS} based on the [CLS] token, which represents a summarized representation of the entire segment. This representation is then passed through a dense layer followed by a sigmoid activation function, producing probability values that a particular schema is present:

$$P(s_j \mid C) = \sigma(Wh_{CLS} + b),$$

where s_j represents a specific schema and σ is the sigmoid activation function. The boundary detection model measures error using categorical cross-entropy, while the schema classification model uses binary cross-entropy (Devlin et al., 2017). Mathematically, the interaction between the two models can be represented as:

1. Chunk detection: $\{T_1, T_2, \ldots, T_n\} \rightarrow \{C_1, C_2, \ldots, C_k\}$
2. Schema classification: $\{C_1, C_2, \ldots, C_k\} \rightarrow \{S_1, S_2, \ldots, S_k\}$

The models are further fine-tuned using the AdamW optimizer with a learning rate of 5e-5, achieving a balance between convergence speed and stability (Kingma & Ba, 2017). Training was conducted over 50 epochs[2] with a batch size[3] of 32, optimizing computational efficiency without compromising model performance. Gradient accumulation[4] was applied to effectively utilize the batch size within the constraints of available GPU memory, ensuring that the models could process a sufficient amount of data per update step. To prevent overfitting[5] and improve generalization, both models incorporated dropout regularization with a dropout rate of 0.18.

5. Model performance

The model's performance was evaluated using a set of metrics covering precision and recall, providing a balanced assessment of its capabilities. On the validation set, the boundary detection model achieved the following results:

- Evaluation Loss: 1.459
- Precision: 0.309
- Recall: 0.347
- F1 Score: 0.327

[2] An epoch represents the complete processing of the entire data group used for training, which means that during each epoch the model sees all the examples from the training set once.

[3] A batch size of 32 means that 32 examples are processed at the same time before the model parameters are adjusted, which allows for work with large datasets in smaller, more easily manageable segments.

[4] Gradient accumulation adds the gradients from several smaller batches before it applies a single update. Thus the model can simulate a greater effective batch size without the need for additional GPU memory.

[5] Dropout regularization randomly excludes a certain percentage of neurons in a network during training – in this instance 10%, to avoid allowing the model to become overdependent on any individual group of links. This improves the overall robustness of the model.

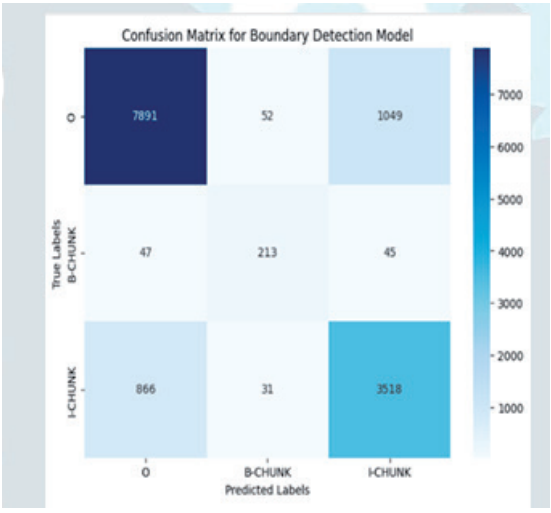
These values indicate a moderate level of performance, with room for improvement in accurately detecting chunk boundaries. The relatively low precision and recall suggest challenges in minimizing false positives and false negatives. The confusion matrix is provided in Appendix 1.

In contrast, the schema classification model demonstrated impressive performance on the validation set, achieving the following results:

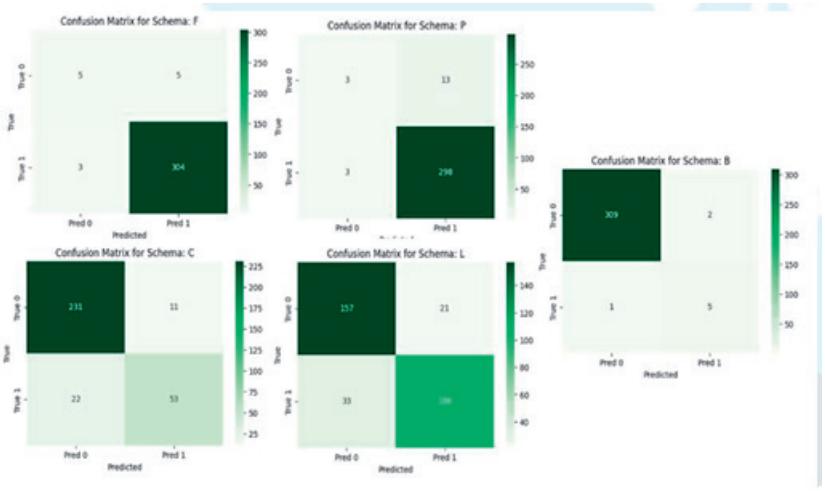
- Evaluation Loss: 0.356
- Precision: 0.936
- Recall: 0.925
- F1 Score: 0.930
- Accuracy: 69.71%

The high levels of precision, recall, and F1 score reflect the model's robust ability to accurately assign multiple relevant schema labels to each text chunk. These metrics indicate the model's expertise in handling multi-label classification tasks, effectively balancing the trade-offs between precision and recall to achieve strong overall performance. Confusion matrices for all five schema categories are provided in Appendix 2.

6. The discussion and further studies



Appendix 1: Confusion Matrix for the Boundary Detection Model



Appendix 2: Confusion Matrices for the Schema Identification Model

These two models are combined as follows: for any new input sequence of tokens, regardless of its length, the segmentation model first extracts meaningful chunks that carry schemas. For example, for the hypothetical input sentence chain:

"He fell into a hole. He moved away from it. Then, he went into a house." the model returns the following annotation:

- **Chunk 1:** he fell into a hole
  Identified schemas: P, F, C
- **Chunk 2:** he moved away from it
  Identified schemas: P, F, L
- **Chunk 3:** then, he went into a house
  Identified schemas: P, F, C

After optimizing the boundary detection and schema classification models, future directions of this research are focused on developing advanced models that integrate scaling modifiers and specifiers, thereby enriching the semantic depth and contextual precision of the entire system. The scaling model is designed to complement the identified schemas with quantitative adjustments, such as positive or negative indicators (+/-), which denote the magnitude and direction of the schema's valence. By integrating these modifiers, the scaling model aims to increase the granularity of schema annotations and to enable subtler interpretations and applications that require precise quantitative data.

At the same time, a specification model has been designed to refine and contextualize existing schemas. This model aims to add modifiers such as "up," "down," "end path," and similar qualifiers, providing schemas with additional layers of semantic information, more detailed and context-aware classification, and flexibility for tasks requiring a high degree of specificity and adaptation.

However, the current architecture shows significant limitations, particularly in the boundary detection component, where the Token-Level F1 Score was 0.3067.

This result indicates moderate accuracy in chunk boundary identification, while the low precision and recall suggest a tendency toward false positives and false negatives. Since this model passes the formed chunks to subsequent processing, these errors can propagate to the next models, including the scaling and specification models. Addressing these challenges requires:

- Extended training regimes: Longer training periods, iterative hyperparameter optimization, and advanced regularization techniques to improve accuracy and generalization capabilities of the models.
- Increasing the dataset: Incorporating more diverse and representative samples in the training set to mitigate overfitting and enhance the model's ability to generalize across different textual contexts.
- Advanced techniques: Using cross-validation and ensemble learning to further strengthen the robustness of the segmentation model.

Furthermore, integrating the scaling and specification models into the existing pipeline requires a cohesive architectural framework that ensures smooth data flow and interoperability between components. This entails developing sophisticated data preprocessing procedures capable of handling additional layers of annotations, such as applying the SCALE schema to other schemas. From an architectural perspective, employing modular and scalable design principles will be crucial to accommodate the growing complexity and interdependencies of the expanded pipeline structure.

References

Antović, M., Jovanović, V. Ž., & Figar, V. (2023). Dynamic schematic complexes: Image schema interaction in music and language cognition reveals a potential for computational affect detection. *Pragmatics & Cognition, 30*(2), 258–295.

Bennett, B., & Cialone, C. (2014). Corpus Guided Sense Cluster Analysis: a methodology for ontology development (with examples from the spatial domain). *Formal Ontology in Information Systems*, *267*, 213–226. https://doi.org/10.3233/978-1-61499-438-1-213

Bird, S. (2006, July). NLTK: the natural language toolkit. In Curran, J. (Ed.) *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions* (pp. 69–72). https://doi.org/10.3115/1225403.1225421

Brownlee, J. (2020, August 17). *Ordinal and one-hot encodings for Categorical Data*. MachineLearningMastery.com. https://machinelearningmastery.com/one-hot-encoding-for categorical-data/ Chardet. PyPI. (n.d.). https://pypi.org/project/chardet/

Clausner, T. C., & Croft, W. (1999). Domains and image schemas. *Cognitive Linguistics, 10*(1), 1–31. https://doi.org/10.1515/cogl.1999.001

Collins, G., & Syme, D. (1995). A theory of finite maps. In E. T. Schubert, P. J. Windley, & J. Alves-Floss (Eds.), *Higher Order Logic Theorem Proving and Its Applications: 8th International Workshop Aspen Grove*, *UT*, *USA*, *September 11–14*, *1995 Proceedings* (pp. 122–137). Berlin: Springer.

Collobert, R., Bengio, S., & Mariethoz, J. (2002). *Torch: a modular machine learning software library*. IDIAP RR 02-46. Lugano: Dalle Molle Institute for Perceptual Intelligence.

Dahlgren, K. (1988). *Naive semantics for natural language understanding*. Norwell/ Dordrecht: Kluwer Academic Publishers.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, May 24). *Bert: Pre training of deep bidirectional Transformers for language understanding*. arXiv.org. https:// arxiv.org/abs/1810.04805

Dodge, E., & Lakoff, G. (2005). Image schemas: From linguistic analysis to neural grounding. In B. Hampe (Ed.), *From Perception to Meaning* (pp. 57–92). Berlin: Mouton De Gruyter. https://doi.org/10.1515/9783110197532.1.57

Evans, V., & Green, M. (2006). *Cognitive linguistics: An introduction*. Edinburgh: Edinburgh University Press.

Gibbs Jr, R. W., Beitel, D. A., Harrington, M., & Sanders, P. E. (1994). Taking a stand on the meanings of stand: Bodily experience as motivation for polysemy. *Journal of Semantics, 11*(4), 231–251.

Gibbs, R. W., & Colston, H. L. (1995). The cognitive psychological reality of image schemas and their transformations. *Cognitive Linguistics, 6*(4), 347–378. https://doi. org/10.1515/cogl.1995.6.4.347

Gromann, D., & Hedblom, M. M. (2017). Kinesthetic mind reader: A method to identify image schemas in natural language. In P. Langley (Ed.), *Advances in Cognitive Systems, 5*, Paper 9. Cognitive Systems Foundation. https://dagmargromann.com/ files/ACS_final_2017.pdf.

Helbig, H. (2014). *Knowledge representation and the semantics of natural language*. Berlin: Springer.

Johnson, M. (1987). *The body in the mind: The bodily basis of meaning, imagination, and reason*. Chicago: University of Chicago Press.

Kapetanios, E., Tatar, D., & Sacarea, C. (2013). *Natural language processing: semantic aspects*. Boca Raton: CRC Press.

Kingma, D. P., & Ba, J. (2017, January 30). *Adam: A method for stochastic optimization*. arXiv.org. https://arxiv.org/abs/1412.6980

Lakoff, G. (1987). *Women, fire, and dangerous things: What categories reveal about the mind*. Chicago: University of Chicago Press.

Lakoff, G., & Johnson, M. (1999). *Philosophy in the flesh: The embodied mind and its challenge to Western thought*. New York City: Basic Books.

Langacker, R. W. (2008). *Cognitive grammar: A basic introduction*. Oxford: Oxford University Press.

Mandler, J. M. (2004). *The foundations of mind: Origins of conceptual thought*. Oxford: Oxford University Press.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of machine Learning research, 12*, 2825–2830.

Van Rossum, G. (2020). *The Python Library Reference*, release 3.8.2. Python Software Foundation.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, June 30). *Attention is all you need*. arXiv.org. https://arxiv.org/abs/1706.03762v4

Wachowiak, L. (2020). Semi-automatic Extraction of Image Schemas from Natural Language. In L. Gschwandtner et al. (Eds.), *Proceedings of the MEi:CogSci Conference 2020* (p. 105). Bratislava: Comenius University.

Wachowiak, L., & Gromann, D. (2022). Systematic analysis of image schemas in natural language through explainable multilingual neural language processing. In N. Calzorali et al. (Eds.), *Proceedings of the 29th International Conference on Computational Linguistics* (pp. 5571–5581). International Committee on Computational Linguistics.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., … Rush, A. M. (2020, July 14). Huggingface's transformers: State-of-the-art natural language processing. Sydney: Association for Computational Linguistics. Retrieved from /https://arxiv.org/abs/1910.03771