

Integrate image2text and text2board workflows

Exported on 12/4/2025 at 11:19:14 EST from Cursor (2.0.77)

User

In Matt's workspace, I have a workflow that is image2text and then text2board, I need to implement this into the simple AAC board UI that i created so that it is one continuous workflow. Essentially, you can open the AAC board, upload an image, or select to type in context of a scenario, then it generates an AAC board. You should take a look at how the text2board works, it has an output that ranks each potential word, I want to implement a slider in the UI that allows you to choose from not much specificity (so probably the like top 10 items) to very much specificity (top 30 items?). Let me know if you have any questions before doing this work. Also, again, I want this as simple as possible. I want to deliver something nice but dont go crazy creating new files.

Cursor

Reviewing the current AAC board UI and the image2text and text2board workflows to understand the structure.

Checking for other relevant files to understand the complete workflow:

Checking the CSV structure to understand the data format:

Implementing the integrated workflow in the AAC board UI:

Implementing the integrated workflow with image upload, text input, and a specificity slider:

""""

AAC Board Generator with Image-to-Text and Text-to-Board workflow.

Features:

- Upload an image or type text description
 - Generate AAC board using BLIP (image captioning) and sentence transformers (text matching)
 - Adjustable specificity slider (10-30 items)
- """"

```

import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import pandas as pd
from PIL import Image, ImageTk
import requests
from io import BytesIO
import torch
from transformers import BlipProcessor, BlipForConditionalGeneration
from sentence_transformers import SentenceTransformer, util
import os

class AACBoard:
    def __init__(self, csv_path):
        """
        Initialize the AAC board interface.

        Args:
            csv_path: Path to the arasaac_synset_mapping CSV file
        """
        self.csv_path = csv_path

        # Load CSV
        print("Loading CSV data...")
        self.df = pd.read_csv(csv_path)
        self.df = self.df.drop_duplicates(subset=['primary_keyword'])

        # Initialize models
        print("Loading models...")
        self.device = "cuda" if torch.cuda.is_available() else "cpu"

        # BLIP for image captioning
        try:
            self.blip_processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
            self.blip_model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base").to(self.device)
            self.blip_loaded = True
        except Exception as e:
            print(f"Warning: Could not load BLIP model: {e}")
            self.blip_loaded = False

        # Sentence transformer for text matching
        try:
            self.sentence_model = SentenceTransformer("all-MiniLM-L6-v2")
            self.sentence_model_loaded = True
        except Exception as e:
            print(f"Warning: Could not load sentence transformer: {e}")
            self.sentence_model_loaded = False

        # Pre-compute keyword embeddings
        if self.sentence_model_loaded:
            print("Pre-computing keyword embeddings...")

```

```
self.df["keyword_emb"] = self.df["primary_keyword"].apply(  
    lambda x: self.sentence_model.encode(x, convert_to_tensor=True))
```

```
# Create main window  
self.root = tk.Tk()  
self.root.title("AAC Board Generator")  
self.root.geometry("1000x700")
```

```
# Variables  
self.current_synset_ids = []  
self.synset_map = {}  
self.input_mode = tk.StringVar(value="text")  
self.specificity = tk.IntVar(value=20)  
self.current_image_path = None
```

```
# Create UI  
self.create_ui()
```

```
def create_ui(self):  
    """Create the user interface."""  
    # Top frame for controls  
    control_frame = ttk.Frame(self.root, padding="10")  
    control_frame.pack(fill=tk.X)  
  
    # Input mode selection  
    mode_frame = ttk.LabelFrame(control_frame, text="Input Type", padding="10")  
    mode_frame.pack(side=tk.LEFT, padx=5)  
  
    ttk.Radiobutton(mode_frame, text="Text Description", variable=self.input_mode,  
                    value="text", command=self.on_mode_change).pack(side=tk.LEFT, padx=5)  
    ttk.Radiobutton(mode_frame, text="Upload Image", variable=self.input_mode,  
                    value="image", command=self.on_mode_change).pack(side=tk.LEFT, padx=5)  
  
    # Text input  
    self.text_frame = ttk.LabelFrame(control_frame, text="Text Description", padding="10")  
    self.text_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)  
  
    self.text_entry = tk.Text(self.text_frame, height=3, width=40)  
    self.text_entry.pack(fill=tk.BOTH, expand=True)  
  
    # Image upload button  
    self.image_frame = ttk.LabelFrame(control_frame, text="Image Upload", padding="10")  
    self.image_button = ttk.Button(self.image_frame, text="Select Image",  
                                   command=self.select_image)  
    self.image_button.pack()  
    self.image_label = ttk.Label(self.image_frame, text="No image selected")  
    self.image_label.pack()  
  
    # Specificity slider  
    slider_frame = ttk.LabelFrame(control_frame, text="Specificity", padding="10")  
    slider_frame.pack(side=tk.LEFT, padx=5)  
  
    ttk.Label(slider_frame, text="Items:").pack()  
    self.slider = ttk.Scale(slider_frame, from_=10, to=30, variable=self.specificity,
```

```

        orient=tk.HORIZONTAL, length=150)
self.slider.pack()
self.slider_label = ttk.Label(slider_frame, text="20")
self.slider_label.pack()
self.specificity.trace('w', lambda *args:
self.slider_label.config(text=str(self.specificity.get()))))

# Generate button
self.generate_button = ttk.Button(control_frame, text="Generate Board",
                                command=self.generate_board)
self.generate_button.pack(side=tk.LEFT, padx=10)

# Status label
self.status_label = ttk.Label(control_frame, text="Ready")
self.status_label.pack(side=tk.LEFT, padx=10)

# Main frame for board
main_frame = ttk.Frame(self.root, padding="10")
main_frame.pack(fill=tk.BOTH, expand=True)

# Create canvas with scrollbar
self.canvas = tk.Canvas(main_frame)
scrollbar = ttk.Scrollbar(main_frame, orient="vertical", command=self.canvas.yview)
self.scrollable_frame = ttk.Frame(self.canvas)

self.scrollable_frame.bind(
    "<Configure>",
    lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all"))
)

self.canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")
self.canvas.configure(yscrollcommand=scrollbar.set)

self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# Bind mousewheel
def on_mousewheel(event):
    self.canvas.yview_scroll(int(-1*(event.delta/120)), "units")
self.canvas.bind_all("<MouseWheel>", on_mousewheel)

# Initial mode
self.on_mode_change()

def on_mode_change(self):
    """Handle input mode change."""
    if self.input_mode.get() == "text":
        self.text_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)
        self.image_frame.pack_forget()
    else:
        self.text_frame.pack_forget()
        self.image_frame.pack(side=tk.LEFT, padx=5)

def select_image(self):

```

```

"""Open file dialog to select image."""
file_path = filedialog.askopenfilename(
    title="Select Image",
    filetypes=[("Image files", "*.jpg *.jpeg *.png *.gif *.bmp")]
)
if file_path:
    self.current_image_path = file_path
    self.image_label.config(text=os.path.basename(file_path))

def caption_image(self, image_path):
    """Generate caption from image using BLIP."""
    if not self.blip_loaded:
        raise Exception("BLIP model not loaded")

    image = Image.open(image_path).convert("RGB")
    inputs = self.blip_processor(images=image, return_tensors="pt").to(self.device)

    with torch.no_grad():
        out = self.blip_model.generate(**inputs, max_length=30, num_beams=5,
repetition_penalty=1.15)

    caption = self.blip_processor.decode(out[0], skip_special_tokens=True).strip()
    return caption

def text_to_synsets(self, text, top_n=20):
    """Convert text to ranked synset IDs using sentence transformers."""
    if not self.sentence_model_loaded:
        raise Exception("Sentence transformer model not loaded")

    # Encode input text
    text_emb = self.sentence_model.encode(text, convert_to_tensor=True)

    # Calculate similarity scores
    self.df["score"] = self.df["keyword_emb"].apply(
        lambda emb: util.cos_sim(text_emb, emb).item())

    # Sort by score and get top N
    df_sorted = self.df.sort_values("score", ascending=False)
    top_results = df_sorted.head(top_n)

    # Extract unique synsets (prefer highest scoring)
    synset_ids = []
    seen_synsets = set()
    for _, row in top_results.iterrows():
        synset = row['synset']
        if synset not in seen_synsets:
            synset_ids.append(synset)
            seen_synsets.add(synset)
            if len(synset_ids) >= top_n:
                break

    return synset_ids

def generate_board(self):

```

```

"""Generate AAC board from input."""
try:
    self.status_label.config(text="Generating...")
    self.root.update()

    # Get text input
    if self.input_mode.get() == "image":
        if not self.current_image_path:
            messagebox.showwarning("No Image", "Please select an image first.")
            return

        if not self.blip_loaded:
            messagebox.showerror("Error", "BLIP model not available. Please install
transformers.")
            return

        # Generate caption from image
        text = self.caption_image(self.current_image_path)
        print(f"Generated caption: {text}")
    else:
        text = self.text_entry.get("1.0", tk.END).strip()
        if not text:
            messagebox.showwarning("No Text", "Please enter a text description.")
            return

    if not self.sentence_model_loaded:
        messagebox.showerror("Error", "Sentence transformer model not available.")
        return

    # Convert text to synset IDs
    top_n = int(self.specificity.get())
    synset_ids = self.text_to_synsets(text, top_n)

    if not synset_ids:
        messagebox.showinfo("No Results", "No matching symbols found.")
        return

    # Update board
    self.current_synset_ids = synset_ids
    self.update_board()

    self.status_label.config(text=f"Generated {len(synset_ids)} symbols")

except Exception as e:
    messagebox.showerror("Error", f"Failed to generate board: {str(e)}")
    self.status_label.config(text="Error")
    print(f"Error: {e}")
    import traceback
    traceback.print_exc()

def update_board(self):
    """Update the board display with current synset IDs."""
    # Clear existing buttons
    for widget in self.scrollable_frame.winfo_children():

```

```

        widget.destroy()

# Build synset map
self.synset_map = {}
for synset in self.current_synset_ids:
    matches = self.df[self.df['synset'] == synset]
    if not matches.empty:
        row = matches.iloc[0]
        self.synset_map[synset] = {
            'image_url': row['image_url'],
            'keyword': row['primary_keyword'],
            'pictogram_id': row['pictogram_id']
        }

# Load and display buttons
self.load_buttons(self.scrollable_frame)

def load_image_from_url(self, url):
    """Load image from URL and resize for button."""
    try:
        response = requests.get(url, timeout=5)
        img = Image.open(BytesIO(response.content))
        img = img.resize((100, 100), Image.Resampling.LANCZOS)
        return ImageTk.PhotoImage(img)
    except Exception as e:
        print(f"Error loading image from {url}: {e}")
        # Return a placeholder
        img = Image.new('RGB', (100, 100), color='gray')
        return ImageTk.PhotoImage(img)

def load_buttons(self, parent):
    """Load and display buttons for each synset."""
    row = 0
    col = 0
    cols_per_row = 4

    for synset_id in self.current_synset_ids:
        if synset_id not in self.synset_map:
            print(f"Warning: Synset {synset_id} not found in CSV")
            continue

        info = self.synset_map[synset_id]

        # Create button frame
        btn_frame = ttk.Frame(parent, padding="5")
        btn_frame.grid(row=row, column=col, padx=5, pady=5)

        # Load image
        photo = self.load_image_from_url(info['image_url'])

        # Create button
        btn = tk.Button(
            btn_frame,
            image=photo,

```

```

        text=info['keyword'],
        compound=tk.TOP,
        width=120,
        height=130,
        command=lambda s=synset_id: self.on_button_click(s)
    )
    btn.image = photo # Keep a reference
    btn.pack()

    # Update grid position
    col += 1
    if col >= cols_per_row:
        col = 0
        row += 1

def on_button_click(self, synset_id):
    """Handle button click."""
    info = self.synset_map.get(synset_id, {})
    keyword = info.get('keyword', synset_id)
    print(f"Clicked: {keyword} ({synset_id})")
    # You can extend this to handle speech synthesis, etc.

def run(self):
    """Start the GUI."""
    self.root.mainloop()

if __name__ == "__main__":
    csv_path = "../Kaitlin's Workspace/arasaac_synset_mapping_20251106_130530.csv"

    board = AACBoard(csv_path)
    board.run()

```

Checking for linting errors and optimizing embedding computation:

Optimizing embedding computation for better performance and memory usage:

"""

AAC Board Generator with Image-to-Text and Text-to-Board workflow.

Features:

- Upload an image or type text description
- Generate AAC board using BLIP (image captioning) and sentence transformers (text matching)
- Adjustable specificity slider (10-30 items)

"""

```

import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import pandas as pd
from PIL import Image, ImageTk
import requests
from io import BytesIO
import torch

```



```
from transformers import BlipProcessor, BlipForConditionalGeneration
from sentence_transformers import SentenceTransformer, util
import os
```

```
class AACBoard:
```

```
    def __init__(self, csv_path):
```

```
        """
```

```
        Initialize the AAC board interface.
```

```
        Args:
```

```
            csv_path: Path to the arasaac_synset_mapping CSV file
```

```
        """
```

```
        self.csv_path = csv_path
```

```
        # Load CSV
```

```
        print("Loading CSV data...")
```

```
        self.df = pd.read_csv(csv_path)
```

```
        self.df = self.df.drop_duplicates(subset=['primary_keyword'])
```

```
        # Initialize models
```

```
        print("Loading models...")
```

```
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
        # BLIP for image captioning
```

```
        try:
```

```
            self.blip_processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-
base")
```

```
            self.blip_model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-
captioning-base").to(self.device)
```

```
            self.blip_loaded = True
```

```
        except Exception as e:
```

```
            print(f"Warning: Could not load BLIP model: {e}")
```

```
            self.blip_loaded = False
```

```
        # Sentence transformer for text matching
```

```
        try:
```

```
            self.sentence_model = SentenceTransformer("all-MiniLM-L6-v2")
```

```
            self.sentence_model_loaded = True
```

```
        except Exception as e:
```

```
            print(f"Warning: Could not load sentence transformer: {e}")
```

```
            self.sentence_model_loaded = False
```

```
        # Pre-compute keyword embeddings (store as numpy arrays for efficiency)
```

```
        if self.sentence_model_loaded:
```

```
            print("Pre-computing keyword embeddings...")
```

```
            keywords = self.df["primary_keyword"].tolist()
```

```
            # Compute embeddings in batches for efficiency
```

```
            embeddings = self.sentence_model.encode(keywords, show_progress_bar=True,
convert_to_numpy=True)
```

```
            self.df["keyword_emb"] = [emb for emb in embeddings]
```

```
        # Create main window
```

```
        self.root = tk.Tk()
```

```
self.root.title("AAC Board Generator")
self.root.geometry("1000x700")
```

```
# Variables
```

```
self.current_synset_ids = []
self.synset_map = {}
self.input_mode = tk.StringVar(value="text")
self.specificity = tk.IntVar(value=20)
self.current_image_path = None
```

```
# Create UI
```

```
self.create_ui()
```

```
def create_ui(self):
```

```
    """Create the user interface."""
```

```
    # Top frame for controls
```

```
    control_frame = ttk.Frame(self.root, padding="10")
    control_frame.pack(fill=tk.X)
```

```
    # Input mode selection
```

```
    mode_frame = ttk.LabelFrame(control_frame, text="Input Type", padding="10")
    mode_frame.pack(side=tk.LEFT, padx=5)
```

```
    ttk.Radiobutton(mode_frame, text="Text Description", variable=self.input_mode,
                    value="text", command=self.on_mode_change).pack(side=tk.LEFT, padx=5)
```

```
    ttk.Radiobutton(mode_frame, text="Upload Image", variable=self.input_mode,
                    value="image", command=self.on_mode_change).pack(side=tk.LEFT, padx=5)
```

```
    # Text input
```

```
    self.text_frame = ttk.LabelFrame(control_frame, text="Text Description", padding="10")
    self.text_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)
```

```
    self.text_entry = tk.Text(self.text_frame, height=3, width=40)
    self.text_entry.pack(fill=tk.BOTH, expand=True)
```

```
    # Image upload button
```

```
    self.image_frame = ttk.LabelFrame(control_frame, text="Image Upload", padding="10")
    self.image_button = tk.Button(self.image_frame, text="Select Image",
                                  command=self.select_image)
```

```
    self.image_button.pack()
```

```
    self.image_label = tk.Label(self.image_frame, text="No image selected")
```

```
    self.image_label.pack()
```

```
    # Specificity slider
```

```
    slider_frame = ttk.LabelFrame(control_frame, text="Specificity", padding="10")
    slider_frame.pack(side=tk.LEFT, padx=5)
```

```
    ttk.Label(slider_frame, text="Items:").pack()
```

```
    self.slider = tk.Scale(slider_frame, from_=10, to=30, variable=self.specificity,
                          orient=tk.HORIZONTAL, length=150)
```

```
    self.slider.pack()
```

```
    self.slider_label = tk.Label(slider_frame, text="20")
```

```
    self.slider_label.pack()
```

```
    self.specificity.trace('w', lambda *args:
```

```

self.slider_label.config(text=str(self.specificity.get()))

# Generate button
self.generate_button = ttk.Button(control_frame, text="Generate Board",
                                   command=self.generate_board)
self.generate_button.pack(side=tk.LEFT, padx=10)

# Status label
self.status_label = ttk.Label(control_frame, text="Ready")
self.status_label.pack(side=tk.LEFT, padx=10)

# Main frame for board
main_frame = ttk.Frame(self.root, padding="10")
main_frame.pack(fill=tk.BOTH, expand=True)

# Create canvas with scrollbar
self.canvas = tk.Canvas(main_frame)
scrollbar = ttk.Scrollbar(main_frame, orient="vertical", command=self.canvas.yview)
self.scrollable_frame = ttk.Frame(self.canvas)

self.scrollable_frame.bind(
    "<Configure>",
    lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all"))
)

self.canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")
self.canvas.configure(yscrollcommand=scrollbar.set)

self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# Bind mousewheel
def on_mousewheel(event):
    self.canvas.yview_scroll(int(-1*(event.delta/120)), "units")
self.canvas.bind_all("<MouseWheel>", on_mousewheel)

# Initial mode
self.on_mode_change()

def on_mode_change(self):
    """Handle input mode change."""
    if self.input_mode.get() == "text":
        self.text_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)
        self.image_frame.pack_forget()
    else:
        self.text_frame.pack_forget()
        self.image_frame.pack(side=tk.LEFT, padx=5)

def select_image(self):
    """Open file dialog to select image."""
    file_path = filedialog.askopenfilename(
        title="Select Image",
        filetypes=[("Image files", "*.jpg *.jpeg *.png *.gif *.bmp")]
    )

```

```

if file_path:
    self.current_image_path = file_path
    self.image_label.config(text=os.path.basename(file_path))

def caption_image(self, image_path):
    """Generate caption from image using BLIP."""
    if not self.blip_loaded:
        raise Exception("BLIP model not loaded")

    image = Image.open(image_path).convert("RGB")
    inputs = self.blip_processor(images=image, return_tensors="pt").to(self.device)

    with torch.no_grad():
        out = self.blip_model.generate(**inputs, max_length=30, num_beams=5,
        repetition_penalty=1.15)

    caption = self.blip_processor.decode(out[0], skip_special_tokens=True).strip()
    return caption

def text_to_synsets(self, text, top_n=20):
    """Convert text to ranked synset IDs using sentence transformers."""
    if not self.sentence_model_loaded:
        raise Exception("Sentence transformer model not loaded")

    # Encode input text
    text_emb = self.sentence_model.encode(text, convert_to_tensor=True)

    # Calculate similarity scores
    self.df["score"] = self.df["keyword_emb"].apply(
        lambda emb: util.cos_sim(text_emb, emb).item())

    # Sort by score and get top N
    df_sorted = self.df.sort_values("score", ascending=False)
    top_results = df_sorted.head(top_n)

    # Extract unique synsets (prefer highest scoring)
    synset_ids = []
    seen_synsets = set()
    for _, row in top_results.iterrows():
        synset = row['synset']
        if synset not in seen_synsets:
            synset_ids.append(synset)
            seen_synsets.add(synset)
            if len(synset_ids) >= top_n:
                break

    return synset_ids

def generate_board(self):
    """Generate AAC board from input."""
    try:
        self.status_label.config(text="Generating...")
        self.root.update()

```

```

# Get text input
if self.input_mode.get() == "image":
    if not self.current_image_path:
        messagebox.showwarning("No Image", "Please select an image first.")
        return

    if not self.blip_loaded:
        messagebox.showerror("Error", "BLIP model not available. Please install
transformers.")
        return

    # Generate caption from image
    text = self.caption_image(self.current_image_path)
    print(f"Generated caption: {text}")
else:
    text = self.text_entry.get("1.0", tk.END).strip()
    if not text:
        messagebox.showwarning("No Text", "Please enter a text description.")
        return

if not self.sentence_model_loaded:
    messagebox.showerror("Error", "Sentence transformer model not available.")
    return

# Convert text to synset IDs
top_n = int(self.specificity.get())
synset_ids = self.text_to_synsets(text, top_n)

if not synset_ids:
    messagebox.showinfo("No Results", "No matching symbols found.")
    return

# Update board
self.current_synset_ids = synset_ids
self.update_board()

self.status_label.config(text=f"Generated {len(synset_ids)} symbols")

except Exception as e:
    messagebox.showerror("Error", f"Failed to generate board: {str(e)}")
    self.status_label.config(text="Error")
    print(f"Error: {e}")
    import traceback
    traceback.print_exc()

def update_board(self):
    """Update the board display with current synset IDs."""
    # Clear existing buttons
    for widget in self.scrollable_frame.winfo_children():
        widget.destroy()

    # Build synset map
    self.synset_map = {}
    for synset in self.current_synset_ids:

```

```

matches = self.df[self.df['synset'] == synset]
if not matches.empty:
    row = matches.iloc[0]
    self.synset_map[synset] = {
        'image_url': row['image_url'],
        'keyword': row['primary_keyword'],
        'pictogram_id': row['pictogram_id']
    }

# Load and display buttons
self.load_buttons(self.scrollable_frame)

def load_image_from_url(self, url):
    """Load image from URL and resize for button."""
    try:
        response = requests.get(url, timeout=5)
        img = Image.open(BytesIO(response.content))
        img = img.resize((100, 100), Image.Resampling.LANCZOS)
        return ImageTk.PhotoImage(img)
    except Exception as e:
        print(f"Error loading image from {url}: {e}")
        # Return a placeholder
        img = Image.new('RGB', (100, 100), color='gray')
        return ImageTk.PhotoImage(img)

def load_buttons(self, parent):
    """Load and display buttons for each synset."""
    row = 0
    col = 0
    cols_per_row = 4

    for synset_id in self.current_synset_ids:
        if synset_id not in self.synset_map:
            print(f"Warning: Synset {synset_id} not found in CSV")
            continue

        info = self.synset_map[synset_id]

        # Create button frame
        btn_frame = ttk.Frame(parent, padding="5")
        btn_frame.grid(row=row, column=col, padx=5, pady=5)

        # Load image
        photo = self.load_image_from_url(info['image_url'])

        # Create button
        btn = tk.Button(
            btn_frame,
            image=photo,
            text=info['keyword'],
            compound=tk.TOP,
            width=120,
            height=130,
            command=lambda s=synset_id: self.on_button_click(s)

```

```

    )
    btn.image = photo # Keep a reference
    btn.pack()

    # Update grid position
    col += 1
    if col >= cols_per_row:
        col = 0
        row += 1

def on_button_click(self, synset_id):
    """Handle button click."""
    info = self.synset_map.get(synset_id, {})
    keyword = info.get('keyword', synset_id)
    print(f"Clicked: {keyword} ({synset_id})")
    # You can extend this to handle speech synthesis, etc.

def run(self):
    """Start the GUI."""
    self.root.mainloop()

if __name__ == "__main__":
    csv_path = "../Kaitlin's Workspace/arasaac_synset_mapping_20251106_130530.csv"

    board = AACBoard(csv_path)
    board.run()

```

Updating the similarity calculation to work with numpy arrays:

```

"""

```

AAC Board Generator with Image-to-Text and Text-to-Board workflow.

Features:

- Upload an image or type text description
- Generate AAC board using BLIP (image captioning) and sentence transformers (text matching)
- Adjustable specificity slider (10-30 items)

```

"""

```

```

import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import pandas as pd
from PIL import Image, ImageTk
import requests
from io import BytesIO
import torch
from transformers import BlipProcessor, BlipForConditionalGeneration
from sentence_transformers import SentenceTransformer, util
import os

```

```

class AACBoard:
    def __init__(self, csv_path):
        """

```

Initialize the AAC board interface.

Args:

csv_path: Path to the arasaac_synset_mapping CSV file

```
"""
self.csv_path = csv_path

# Load CSV
print("Loading CSV data...")
self.df = pd.read_csv(csv_path)
self.df = self.df.drop_duplicates(subset=['primary_keyword'])

# Initialize models
print("Loading models...")
self.device = "cuda" if torch.cuda.is_available() else "cpu"

# BLIP for image captioning
try:
    self.blip_processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-
base")
    self.blip_model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-
captioning-base").to(self.device)
    self.blip_loaded = True
except Exception as e:
    print(f"Warning: Could not load BLIP model: {e}")
    self.blip_loaded = False

# Sentence transformer for text matching
try:
    self.sentence_model = SentenceTransformer("all-MiniLM-L6-v2")
    self.sentence_model_loaded = True
except Exception as e:
    print(f"Warning: Could not load sentence transformer: {e}")
    self.sentence_model_loaded = False

# Pre-compute keyword embeddings (store as numpy arrays for efficiency)
if self.sentence_model_loaded:
    print("Pre-computing keyword embeddings...")
    keywords = self.df["primary_keyword"].tolist()
    # Compute embeddings in batches for efficiency
    embeddings = self.sentence_model.encode(keywords, show_progress_bar=True,
convert_to_numpy=True)
    self.df["keyword_emb"] = [emb for emb in embeddings]

# Create main window
self.root = tk.Tk()
self.root.title("AAC Board Generator")
self.root.geometry("1000x700")

# Variables
self.current_synset_ids = []
self.synset_map = {}
self.input_mode = tk.StringVar(value="text")
self.specificity = tk.IntVar(value=20)
```



```

self.current_image_path = None

# Create UI
self.create_ui()

def create_ui(self):
    """Create the user interface."""
    # Top frame for controls
    control_frame = ttk.Frame(self.root, padding="10")
    control_frame.pack(fill=tk.X)

    # Input mode selection
    mode_frame = ttk.LabelFrame(control_frame, text="Input Type", padding="10")
    mode_frame.pack(side=tk.LEFT, padx=5)

    ttk.Radiobutton(mode_frame, text="Text Description", variable=self.input_mode,
                    value="text", command=self.on_mode_change).pack(side=tk.LEFT, padx=5)
    ttk.Radiobutton(mode_frame, text="Upload Image", variable=self.input_mode,
                    value="image", command=self.on_mode_change).pack(side=tk.LEFT, padx=5)

    # Text input
    self.text_frame = ttk.LabelFrame(control_frame, text="Text Description", padding="10")
    self.text_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)

    self.text_entry = tk.Text(self.text_frame, height=3, width=40)
    self.text_entry.pack(fill=tk.BOTH, expand=True)

    # Image upload button
    self.image_frame = ttk.LabelFrame(control_frame, text="Image Upload", padding="10")
    self.image_button = ttk.Button(self.image_frame, text="Select Image",
                                   command=self.select_image)
    self.image_button.pack()
    self.image_label = ttk.Label(self.image_frame, text="No image selected")
    self.image_label.pack()

    # Specificity slider
    slider_frame = ttk.LabelFrame(control_frame, text="Specificity", padding="10")
    slider_frame.pack(side=tk.LEFT, padx=5)

    ttk.Label(slider_frame, text="Items:").pack()
    self.slider = ttk.Scale(slider_frame, from_=10, to=30, variable=self.specificity,
                           orient=tk.HORIZONTAL, length=150)
    self.slider.pack()
    self.slider_label = ttk.Label(slider_frame, text="20")
    self.slider_label.pack()
    self.specificity.trace('w', lambda *args:
self.slider_label.config(text=str(self.specificity.get()))))

    # Generate button
    self.generate_button = ttk.Button(control_frame, text="Generate Board",
                                      command=self.generate_board)
    self.generate_button.pack(side=tk.LEFT, padx=10)

    # Status label

```

```

self.status_label = ttk.Label(control_frame, text="Ready")
self.status_label.pack(side=tk.LEFT, padx=10)

# Main frame for board
main_frame = ttk.Frame(self.root, padding="10")
main_frame.pack(fill=tk.BOTH, expand=True)

# Create canvas with scrollbar
self.canvas = tk.Canvas(main_frame)
scrollbar = ttk.Scrollbar(main_frame, orient="vertical", command=self.canvas.yview)
self.scrollable_frame = ttk.Frame(self.canvas)

self.scrollable_frame.bind(
    "<Configure>",
    lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all"))
)

self.canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")
self.canvas.configure(yscrollcommand=scrollbar.set)

self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# Bind mousewheel
def on_mousewheel(event):
    self.canvas.yview_scroll(int(-1*(event.delta/120)), "units")
self.canvas.bind_all("<MouseWheel>", on_mousewheel)

# Initial mode
self.on_mode_change()

def on_mode_change(self):
    """Handle input mode change."""
    if self.input_mode.get() == "text":
        self.text_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)
        self.image_frame.pack_forget()
    else:
        self.text_frame.pack_forget()
        self.image_frame.pack(side=tk.LEFT, padx=5)

def select_image(self):
    """Open file dialog to select image."""
    file_path = filedialog.askopenfilename(
        title="Select Image",
        filetypes=[("Image files", "*.jpg *.jpeg *.png *.gif *.bmp")]
    )
    if file_path:
        self.current_image_path = file_path
        self.image_label.config(text=os.path.basename(file_path))

def caption_image(self, image_path):
    """Generate caption from image using BLIP."""
    if not self.blip_loaded:
        raise Exception("BLIP model not loaded")

```

```

image = Image.open(image_path).convert("RGB")
inputs = self.blip_processor(images=image, return_tensors="pt").to(self.device)

with torch.no_grad():
    out = self.blip_model.generate(**inputs, max_length=30, num_beams=5,
repetition_penalty=1.15)

caption = self.blip_processor.decode(out[0], skip_special_tokens=True).strip()
return caption

def text_to_synsets(self, text, top_n=20):
    """Convert text to ranked synset IDs using sentence transformers."""
    if not self.sentence_model_loaded:
        raise Exception("Sentence transformer model not loaded")

    # Encode input text
    text_emb = self.sentence_model.encode(text, convert_to_numpy=True)

    # Calculate similarity scores
    import numpy as np
    self.df["score"] = self.df["keyword_emb"].apply(
        lambda emb: np.dot(text_emb, emb) / (np.linalg.norm(text_emb) * np.linalg.norm(emb)))

    # Sort by score and get top N
    df_sorted = self.df.sort_values("score", ascending=False)
    top_results = df_sorted.head(top_n)

    # Extract unique synsets (prefer highest scoring)
    synset_ids = []
    seen_synsets = set()
    for _, row in top_results.iterrows():
        synset = row['synset']
        if synset not in seen_synsets:
            synset_ids.append(synset)
            seen_synsets.add(synset)
            if len(synset_ids) >= top_n:
                break

    return synset_ids

def generate_board(self):
    """Generate AAC board from input."""
    try:
        self.status_label.config(text="Generating...")
        self.root.update()

        # Get text input
        if self.input_mode.get() == "image":
            if not self.current_image_path:
                messagebox.showwarning("No Image", "Please select an image first.")
                return

            if not self.blip_loaded:

```

```

        messagebox.showerror("Error", "BLIP model not available. Please install
transformers.")
        return

    # Generate caption from image
    text = self.caption_image(self.current_image_path)
    print(f"Generated caption: {text}")
else:
    text = self.text_entry.get("1.0", tk.END).strip()
    if not text:
        messagebox.showwarning("No Text", "Please enter a text description.")
        return

if not self.sentence_model_loaded:
    messagebox.showerror("Error", "Sentence transformer model not available.")
    return

# Convert text to synset IDs
top_n = int(self.specificity.get())
synset_ids = self.text_to_synsets(text, top_n)

if not synset_ids:
    messagebox.showinfo("No Results", "No matching symbols found.")
    return

# Update board
self.current_synset_ids = synset_ids
self.update_board()

self.status_label.config(text=f"Generated {len(synset_ids)} symbols")

except Exception as e:
    messagebox.showerror("Error", f"Failed to generate board: {str(e)}")
    self.status_label.config(text="Error")
    print(f"Error: {e}")
    import traceback
    traceback.print_exc()

def update_board(self):
    """Update the board display with current synset IDs."""
    # Clear existing buttons
    for widget in self.scrollable_frame.winfo_children():
        widget.destroy()

    # Build synset map
    self.synset_map = {}
    for synset in self.current_synset_ids:
        matches = self.df[self.df['synset'] == synset]
        if not matches.empty:
            row = matches.iloc[0]
            self.synset_map[synset] = {
                'image_url': row['image_url'],
                'keyword': row['primary_keyword'],
                'pictogram_id': row['pictogram_id']
            }

```

```
}
```

```
# Load and display buttons
```

```
self.load_buttons(self.scrollable_frame)
```

```
def load_image_from_url(self, url):
```

```
    """Load image from URL and resize for button."""
```

```
    try:
```

```
        response = requests.get(url, timeout=5)
```

```
        img = Image.open(BytesIO(response.content))
```

```
        img = img.resize((100, 100), Image.Resampling.LANCZOS)
```

```
        return ImageTk.PhotoImage(img)
```

```
    except Exception as e:
```

```
        print(f"Error loading image from {url}: {e}")
```

```
        # Return a placeholder
```

```
        img = Image.new('RGB', (100, 100), color='gray')
```

```
        return ImageTk.PhotoImage(img)
```

```
def load_buttons(self, parent):
```

```
    """Load and display buttons for each synset."""
```

```
    row = 0
```

```
    col = 0
```

```
    cols_per_row = 4
```

```
    for synset_id in self.current_synset_ids:
```

```
        if synset_id not in self.synset_map:
```

```
            print(f"Warning: Synset {synset_id} not found in CSV")
```

```
            continue
```

```
        info = self.synset_map[synset_id]
```

```
        # Create button frame
```

```
        btn_frame = ttk.Frame(parent, padding="5")
```

```
        btn_frame.grid(row=row, column=col, padx=5, pady=5)
```

```
        # Load image
```

```
        photo = self.load_image_from_url(info['image_url'])
```

```
        # Create button
```

```
        btn = tk.Button(
```

```
            btn_frame,
```

```
            image=photo,
```

```
            text=info['keyword'],
```

```
            compound=tk.TOP,
```

```
            width=120,
```

```
            height=130,
```

```
            command=lambda s=synset_id: self.on_button_click(s)
```

```
        )
```

```
        btn.image = photo # Keep a reference
```

```
        btn.pack()
```

```
        # Update grid position
```

```
        col += 1
```

```
        if col >= cols_per_row:
```

```
col = 0
row += 1
```

```
def on_button_click(self, synset_id):
    """Handle button click."""
    info = self.synset_map.get(synset_id, {})
    keyword = info.get('keyword', synset_id)
    print(f"Clicked: {keyword} ({synset_id})")
    # You can extend this to handle speech synthesis, etc.
```

```
def run(self):
    """Start the GUI."""
    self.root.mainloop()
```

```
if __name__ == "__main__":
    csv_path = "../Kaitlin's Workspace/arasaac_synset_mapping_20251106_130530.csv"
```

```
    board = AACBoard(csv_path)
    board.run()
```

Moving the numpy import to the top of the file:

```
"""
```

AAC Board Generator with Image-to-Text and Text-to-Board workflow.

Features:

- Upload an image or type text description
- Generate AAC board using BLIP (image captioning) and sentence transformers (text matching)
- Adjustable specificity slider (10-30 items)

```
"""
```

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import pandas as pd
from PIL import Image, ImageTk
import requests
from io import BytesIO
import torch
import numpy as np
from transformers import BlipProcessor, BlipForConditionalGeneration
from sentence_transformers import SentenceTransformer, util
import os
```

```
class AACBoard:
    def __init__(self, csv_path):
        """
        Initialize the AAC board interface.
```

Args:

csv_path: Path to the arasaac_synset_mapping CSV file

```
"""
```

```
self.csv_path = csv_path
```

```

# Load CSV
print("Loading CSV data...")
self.df = pd.read_csv(csv_path)
self.df = self.df.drop_duplicates(subset=['primary_keyword'])

# Initialize models
print("Loading models...")
self.device = "cuda" if torch.cuda.is_available() else "cpu"

# BLIP for image captioning
try:
    self.blip_processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-
base")
    self.blip_model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-
captioning-base").to(self.device)
    self.blip_loaded = True
except Exception as e:
    print(f"Warning: Could not load BLIP model: {e}")
    self.blip_loaded = False

# Sentence transformer for text matching
try:
    self.sentence_model = SentenceTransformer("all-MiniLM-L6-v2")
    self.sentence_model_loaded = True
except Exception as e:
    print(f"Warning: Could not load sentence transformer: {e}")
    self.sentence_model_loaded = False

# Pre-compute keyword embeddings (store as numpy arrays for efficiency)
if self.sentence_model_loaded:
    print("Pre-computing keyword embeddings...")
    keywords = self.df["primary_keyword"].tolist()
    # Compute embeddings in batches for efficiency
    embeddings = self.sentence_model.encode(keywords, show_progress_bar=True,
convert_to_numpy=True)
    self.df["keyword_emb"] = [emb for emb in embeddings]

# Create main window
self.root = tk.Tk()
self.root.title("AAC Board Generator")
self.root.geometry("1000x700")

# Variables
self.current_synset_ids = []
self.synset_map = {}
self.input_mode = tk.StringVar(value="text")
self.specificity = tk.IntVar(value=20)
self.current_image_path = None

# Create UI
self.create_ui()

def create_ui(self):

```

```

"""Create the user interface."""
# Top frame for controls
control_frame = ttk.Frame(self.root, padding="10")
control_frame.pack(fill=tk.X)

# Input mode selection
mode_frame = ttk.LabelFrame(control_frame, text="Input Type", padding="10")
mode_frame.pack(side=tk.LEFT, padx=5)

ttk.Radiobutton(mode_frame, text="Text Description", variable=self.input_mode,
                 value="text", command=self.on_mode_change).pack(side=tk.LEFT, padx=5)
ttk.Radiobutton(mode_frame, text="Upload Image", variable=self.input_mode,
                 value="image", command=self.on_mode_change).pack(side=tk.LEFT, padx=5)

# Text input
self.text_frame = ttk.LabelFrame(control_frame, text="Text Description", padding="10")
self.text_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)

self.text_entry = tk.Text(self.text_frame, height=3, width=40)
self.text_entry.pack(fill=tk.BOTH, expand=True)

# Image upload button
self.image_frame = ttk.LabelFrame(control_frame, text="Image Upload", padding="10")
self.image_button = ttk.Button(self.image_frame, text="Select Image",
                               command=self.select_image)
self.image_button.pack()
self.image_label = ttk.Label(self.image_frame, text="No image selected")
self.image_label.pack()

# Specificity slider
slider_frame = ttk.LabelFrame(control_frame, text="Specificity", padding="10")
slider_frame.pack(side=tk.LEFT, padx=5)

ttk.Label(slider_frame, text="Items:").pack()
self.slider = ttk.Scale(slider_frame, from_=10, to=30, variable=self.specificity,
                        orient=tk.HORIZONTAL, length=150)
self.slider.pack()
self.slider_label = ttk.Label(slider_frame, text="20")
self.slider_label.pack()
self.specificity.trace('w', lambda *args:
self.slider_label.config(text=str(self.specificity.get()))))

# Generate button
self.generate_button = ttk.Button(control_frame, text="Generate Board",
                                  command=self.generate_board)
self.generate_button.pack(side=tk.LEFT, padx=10)

# Status label
self.status_label = ttk.Label(control_frame, text="Ready")
self.status_label.pack(side=tk.LEFT, padx=10)

# Main frame for board
main_frame = ttk.Frame(self.root, padding="10")
main_frame.pack(fill=tk.BOTH, expand=True)

```



```

# Create canvas with scrollbar
self.canvas = tk.Canvas(main_frame)
scrollbar = ttk.Scrollbar(main_frame, orient="vertical", command=self.canvas.yview)
self.scrollable_frame = ttk.Frame(self.canvas)

self.scrollable_frame.bind(
    "<Configure>",
    lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all"))
)

self.canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")
self.canvas.configure(yscrollcommand=scrollbar.set)

self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# Bind mousewheel
def on_mousewheel(event):
    self.canvas.yview_scroll(int(-1*(event.delta/120)), "units")
self.canvas.bind_all("<MouseWheel>", on_mousewheel)

# Initial mode
self.on_mode_change()

def on_mode_change(self):
    """Handle input mode change."""
    if self.input_mode.get() == "text":
        self.text_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)
        self.image_frame.pack_forget()
    else:
        self.text_frame.pack_forget()
        self.image_frame.pack(side=tk.LEFT, padx=5)

def select_image(self):
    """Open file dialog to select image."""
    file_path = filedialog.askopenfilename(
        title="Select Image",
        filetypes=[("Image files", "*.jpg *.jpeg *.png *.gif *.bmp")]
    )
    if file_path:
        self.current_image_path = file_path
        self.image_label.config(text=os.path.basename(file_path))

def caption_image(self, image_path):
    """Generate caption from image using BLIP."""
    if not self.blip_loaded:
        raise Exception("BLIP model not loaded")

    image = Image.open(image_path).convert("RGB")
    inputs = self.blip_processor(images=image, return_tensors="pt").to(self.device)

    with torch.no_grad():
        out = self.blip_model.generate(**inputs, max_length=30, num_beams=5,

```

```
repetition_penalty=1.15)
```

```
caption = self.blip_processor.decode(out[0], skip_special_tokens=True).strip()
return caption
```

```
def text_to_synsets(self, text, top_n=20):
```

```
    """Convert text to ranked synset IDs using sentence transformers."""
```

```
    if not self.sentence_model_loaded:
```

```
        raise Exception("Sentence transformer model not loaded")
```

```
    # Encode input text
```

```
    text_emb = self.sentence_model.encode(text, convert_to_numpy=True)
```

```
    # Calculate similarity scores
```

```
    import numpy as np
```

```
    self.df["score"] = self.df["keyword_emb"].apply(
```

```
        lambda emb: np.dot(text_emb, emb) / (np.linalg.norm(text_emb) * np.linalg.norm(emb)))
```

```
    # Sort by score and get top N
```

```
    df_sorted = self.df.sort_values("score", ascending=False)
```

```
    top_results = df_sorted.head(top_n)
```

```
    # Extract unique synsets (prefer highest scoring)
```

```
    synset_ids = []
```

```
    seen_synsets = set()
```

```
    for _, row in top_results.iterrows():
```

```
        synset = row['synset']
```

```
        if synset not in seen_synsets:
```

```
            synset_ids.append(synset)
```

```
            seen_synsets.add(synset)
```

```
            if len(synset_ids) >= top_n:
```

```
                break
```

```
    return synset_ids
```

```
def generate_board(self):
```

```
    """Generate AAC board from input."""
```

```
    try:
```

```
        self.status_label.config(text="Generating...")
```

```
        self.root.update()
```

```
    # Get text input
```

```
    if self.input_mode.get() == "image":
```

```
        if not self.current_image_path:
```

```
            messagebox.showwarning("No Image", "Please select an image first.")
```

```
            return
```

```
        if not self.blip_loaded:
```

```
            messagebox.showerror("Error", "BLIP model not available. Please install  
transformers.")
```

```
            return
```

```
    # Generate caption from image
```

```
    text = self.caption_image(self.current_image_path)
```

```

        print(f"Generated caption: {text}")
    else:
        text = self.text_entry.get("1.0", tk.END).strip()
        if not text:
            messagebox.showwarning("No Text", "Please enter a text description.")
            return

    if not self.sentence_model_loaded:
        messagebox.showerror("Error", "Sentence transformer model not available.")
        return

    # Convert text to synset IDs
    top_n = int(self.specificity.get())
    synset_ids = self.text_to_synsets(text, top_n)

    if not synset_ids:
        messagebox.showinfo("No Results", "No matching symbols found.")
        return

    # Update board
    self.current_synset_ids = synset_ids
    self.update_board()

    self.status_label.config(text=f"Generated {len(synset_ids)} symbols")

except Exception as e:
    messagebox.showerror("Error", f"Failed to generate board: {str(e)}")
    self.status_label.config(text="Error")
    print(f"Error: {e}")
    import traceback
    traceback.print_exc()

def update_board(self):
    """Update the board display with current synset IDs."""
    # Clear existing buttons
    for widget in self.scrollable_frame.winfo_children():
        widget.destroy()

    # Build synset map
    self.synset_map = {}
    for synset in self.current_synset_ids:
        matches = self.df[self.df['synset'] == synset]
        if not matches.empty:
            row = matches.iloc[0]
            self.synset_map[synset] = {
                'image_url': row['image_url'],
                'keyword': row['primary_keyword'],
                'pictogram_id': row['pictogram_id']
            }

    # Load and display buttons
    self.load_buttons(self.scrollable_frame)

def load_image_from_url(self, url):

```

```

"""Load image from URL and resize for button."""
try:
    response = requests.get(url, timeout=5)
    img = Image.open(BytesIO(response.content))
    img = img.resize((100, 100), Image.Resampling.LANCZOS)
    return ImageTk.PhotoImage(img)
except Exception as e:
    print(f"Error loading image from {url}: {e}")
    # Return a placeholder
    img = Image.new('RGB', (100, 100), color='gray')
    return ImageTk.PhotoImage(img)

def load_buttons(self, parent):
    """Load and display buttons for each synset."""
    row = 0
    col = 0
    cols_per_row = 4

    for synset_id in self.current_synset_ids:
        if synset_id not in self.synset_map:
            print(f"Warning: Synset {synset_id} not found in CSV")
            continue

        info = self.synset_map[synset_id]

        # Create button frame
        btn_frame = ttk.Frame(parent, padding="5")
        btn_frame.grid(row=row, column=col, padx=5, pady=5)

        # Load image
        photo = self.load_image_from_url(info['image_url'])

        # Create button
        btn = tk.Button(
            btn_frame,
            image=photo,
            text=info['keyword'],
            compound=tk.TOP,
            width=120,
            height=130,
            command=lambda s=synset_id: self.on_button_click(s)
        )
        btn.image = photo # Keep a reference
        btn.pack()

        # Update grid position
        col += 1
        if col >= cols_per_row:
            col = 0
            row += 1

    def on_button_click(self, synset_id):
        """Handle button click."""
        info = self.synset_map.get(synset_id, {})

```

```
keyword = info.get('keyword', synset_id)
print(f"Clicked: {keyword} ({synset_id})")
# You can extend this to handle speech synthesis, etc.
```

```
def run(self):
    """Start the GUI."""
    self.root.mainloop()
```

```
if __name__ == "__main__":
    csv_path = "../Kaitlin's Workspace/arasaac_synset_mapping_20251106_130530.csv"

    board = AACBoard(csv_path)
    board.run()
"""
```

AAC Board Generator with Image-to-Text and Text-to-Board workflow.

Features:

- Upload an image or type text description
- Generate AAC board using BLIP (image captioning) and sentence transformers (text matching)
- Adjustable specificity slider (10-30 items)

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import pandas as pd
from PIL import Image, ImageTk
import requests
from io import BytesIO
import torch
import numpy as np
from transformers import BlipProcessor, BlipForConditionalGeneration
from sentence_transformers import SentenceTransformer, util
import os
```

```
class AACBoard:
    def __init__(self, csv_path):
        """
        Initialize the AAC board interface.

        Args:
            csv_path: Path to the arasaac_synset_mapping CSV file
        """
        self.csv_path = csv_path

        # Load CSV
        print("Loading CSV data...")
        self.df = pd.read_csv(csv_path)
        self.df = self.df.drop_duplicates(subset=['primary_keyword'])

        # Initialize models
        print("Loading models...")
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
```

```

# BLIP for image captioning
try:
    self.blip_processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-
base")
    self.blip_model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-
captioning-base").to(self.device)
    self.blip_loaded = True
except Exception as e:
    print(f"Warning: Could not load BLIP model: {e}")
    self.blip_loaded = False

# Sentence transformer for text matching
try:
    self.sentence_model = SentenceTransformer("all-MiniLM-L6-v2")
    self.sentence_model_loaded = True
except Exception as e:
    print(f"Warning: Could not load sentence transformer: {e}")
    self.sentence_model_loaded = False

# Pre-compute keyword embeddings (store as numpy arrays for efficiency)
if self.sentence_model_loaded:
    print("Pre-computing keyword embeddings...")
    keywords = self.df["primary_keyword"].tolist()
    # Compute embeddings in batches for efficiency
    embeddings = self.sentence_model.encode(keywords, show_progress_bar=True,
convert_to_numpy=True)
    self.df["keyword_emb"] = [emb for emb in embeddings]

# Create main window
self.root = tk.Tk()
self.root.title("AAC Board Generator")
self.root.geometry("1000x700")

# Variables
self.current_synset_ids = []
self.synset_map = {}
self.input_mode = tk.StringVar(value="text")
self.specificity = tk.IntVar(value=20)
self.current_image_path = None

# Create UI
self.create_ui()

def create_ui(self):
    """Create the user interface."""
    # Top frame for controls
    control_frame = ttk.Frame(self.root, padding="10")
    control_frame.pack(fill=tk.X)

    # Input mode selection
    mode_frame = ttk.LabelFrame(control_frame, text="Input Type", padding="10")
    mode_frame.pack(side=tk.LEFT, padx=5)

```

```

ttk.Radiobutton(mode_frame, text="Text Description", variable=self.input_mode,
                value="text", command=self.on_mode_change).pack(side=tk.LEFT, padx=5)
ttk.Radiobutton(mode_frame, text="Upload Image", variable=self.input_mode,
                value="image", command=self.on_mode_change).pack(side=tk.LEFT, padx=5)

# Text input
self.text_frame = ttk.LabelFrame(control_frame, text="Text Description", padding="10")
self.text_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)

self.text_entry = tk.Text(self.text_frame, height=3, width=40)
self.text_entry.pack(fill=tk.BOTH, expand=True)

# Image upload button
self.image_frame = ttk.LabelFrame(control_frame, text="Image Upload", padding="10")
self.image_button = ttk.Button(self.image_frame, text="Select Image",
                               command=self.select_image)
self.image_button.pack()
self.image_label = ttk.Label(self.image_frame, text="No image selected")
self.image_label.pack()

# Specificity slider
slider_frame = ttk.LabelFrame(control_frame, text="Specificity", padding="10")
slider_frame.pack(side=tk.LEFT, padx=5)

ttk.Label(slider_frame, text="Items:").pack()
self.slider = tk.Scale(slider_frame, from_=10, to=30, variable=self.specificity,
                      orient=tk.HORIZONTAL, length=150)
self.slider.pack()
self.slider_label = ttk.Label(slider_frame, text="20")
self.slider_label.pack()
self.specificity.trace('w', lambda *args:
self.slider_label.config(text=str(self.specificity.get()))))

# Generate button
self.generate_button = ttk.Button(control_frame, text="Generate Board",
                                  command=self.generate_board)
self.generate_button.pack(side=tk.LEFT, padx=10)

# Status label
self.status_label = ttk.Label(control_frame, text="Ready")
self.status_label.pack(side=tk.LEFT, padx=10)

# Main frame for board
main_frame = ttk.Frame(self.root, padding="10")
main_frame.pack(fill=tk.BOTH, expand=True)

# Create canvas with scrollbar
self.canvas = tk.Canvas(main_frame)
scrollbar = ttk.Scrollbar(main_frame, orient="vertical", command=self.canvas.yview)
self.scrollable_frame = ttk.Frame(self.canvas)

self.scrollable_frame.bind(
    "<Configure>",
    lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all")))

```

```

)

self.canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")
self.canvas.configure(yscrollcommand=scrollbar.set)

self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# Bind mousewheel
def on_mousewheel(event):
    self.canvas.yview_scroll(int(-1*(event.delta/120)), "units")
self.canvas.bind_all("<MouseWheel>", on_mousewheel)

# Initial mode
self.on_mode_change()

def on_mode_change(self):
    """Handle input mode change."""
    if self.input_mode.get() == "text":
        self.text_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)
        self.image_frame.pack_forget()
    else:
        self.text_frame.pack_forget()
        self.image_frame.pack(side=tk.LEFT, padx=5)

def select_image(self):
    """Open file dialog to select image."""
    file_path = filedialog.askopenfilename(
        title="Select Image",
        filetypes=[("Image files", "*.jpg *.jpeg *.png *.gif *.bmp")]
    )
    if file_path:
        self.current_image_path = file_path
        self.image_label.config(text=os.path.basename(file_path))

def caption_image(self, image_path):
    """Generate caption from image using BLIP."""
    if not self.blip_loaded:
        raise Exception("BLIP model not loaded")

    image = Image.open(image_path).convert("RGB")
    inputs = self.blip_processor(images=image, return_tensors="pt").to(self.device)

    with torch.no_grad():
        out = self.blip_model.generate(**inputs, max_length=30, num_beams=5,
        repetition_penalty=1.15)

    caption = self.blip_processor.decode(out[0], skip_special_tokens=True).strip()
    return caption

def text_to_synsets(self, text, top_n=20):
    """Convert text to ranked synset IDs using sentence transformers."""
    if not self.sentence_model_loaded:
        raise Exception("Sentence transformer model not loaded")

```



```

# Encode input text
text_emb = self.sentence_model.encode(text, convert_to_numpy=True)

# Calculate similarity scores (cosine similarity)
self.df["score"] = self.df["keyword_emb"].apply(
    lambda emb: np.dot(text_emb, emb) / (np.linalg.norm(text_emb) * np.linalg.norm(emb)))

# Sort by score and get top N
df_sorted = self.df.sort_values("score", ascending=False)
top_results = df_sorted.head(top_n)

# Extract unique synsets (prefer highest scoring)
synset_ids = []
seen_synsets = set()
for _, row in top_results.iterrows():
    synset = row['synset']
    if synset not in seen_synsets:
        synset_ids.append(synset)
        seen_synsets.add(synset)
    if len(synset_ids) >= top_n:
        break

return synset_ids

def generate_board(self):
    """Generate AAC board from input."""
    try:
        self.status_label.config(text="Generating...")
        self.root.update()

        # Get text input
        if self.input_mode.get() == "image":
            if not self.current_image_path:
                messagebox.showwarning("No Image", "Please select an image first.")
                return

            if not self.blip_loaded:
                messagebox.showerror("Error", "BLIP model not available. Please install transformers.")
                return

            # Generate caption from image
            text = self.caption_image(self.current_image_path)
            print(f"Generated caption: {text}")
        else:
            text = self.text_entry.get("1.0", tk.END).strip()
            if not text:
                messagebox.showwarning("No Text", "Please enter a text description.")
                return

        if not self.sentence_model_loaded:
            messagebox.showerror("Error", "Sentence transformer model not available.")
            return

```

```

# Convert text to synset IDs
top_n = int(self.specificity.get())
synset_ids = self.text_to_synsets(text, top_n)

if not synset_ids:
    messagebox.showinfo("No Results", "No matching symbols found.")
    return

# Update board
self.current_synset_ids = synset_ids
self.update_board()

self.status_label.config(text=f"Generated {len(synset_ids)} symbols")

except Exception as e:
    messagebox.showerror("Error", f"Failed to generate board: {str(e)}")
    self.status_label.config(text="Error")
    print(f"Error: {e}")
    import traceback
    traceback.print_exc()

def update_board(self):
    """Update the board display with current synset IDs."""
    # Clear existing buttons
    for widget in self.scrollable_frame.winfo_children():
        widget.destroy()

    # Build synset map
    self.synset_map = {}
    for synset in self.current_synset_ids:
        matches = self.df[self.df['synset'] == synset]
        if not matches.empty:
            row = matches.iloc[0]
            self.synset_map[synset] = {
                'image_url': row['image_url'],
                'keyword': row['primary_keyword'],
                'pictogram_id': row['pictogram_id']
            }

    # Load and display buttons
    self.load_buttons(self.scrollable_frame)

def load_image_from_url(self, url):
    """Load image from URL and resize for button."""
    try:
        response = requests.get(url, timeout=5)
        img = Image.open(BytesIO(response.content))
        img = img.resize((100, 100), Image.Resampling.LANCZOS)
        return ImageTk.PhotoImage(img)
    except Exception as e:
        print(f"Error loading image from {url}: {e}")
        # Return a placeholder
        img = Image.new('RGB', (100, 100), color='gray')

```

```

    return ImageTk.PhotoImage(img)

def load_buttons(self, parent):
    """Load and display buttons for each synset."""
    row = 0
    col = 0
    cols_per_row = 4

    for synset_id in self.current_synset_ids:
        if synset_id not in self.synset_map:
            print(f"Warning: Synset {synset_id} not found in CSV")
            continue

        info = self.synset_map[synset_id]

        # Create button frame
        btn_frame = ttk.Frame(parent, padding="5")
        btn_frame.grid(row=row, column=col, padx=5, pady=5)

        # Load image
        photo = self.load_image_from_url(info['image_url'])

        # Create button
        btn = tk.Button(
            btn_frame,
            image=photo,
            text=info['keyword'],
            compound=tk.TOP,
            width=120,
            height=130,
            command=lambda s=synset_id: self.on_button_click(s)
        )
        btn.image = photo # Keep a reference
        btn.pack()

        # Update grid position
        col += 1
        if col >= cols_per_row:
            col = 0
            row += 1

    def on_button_click(self, synset_id):
        """Handle button click."""
        info = self.synset_map.get(synset_id, {})
        keyword = info.get('keyword', synset_id)
        print(f"Clicked: {keyword} ({synset_id})")
        # You can extend this to handle speech synthesis, etc.

    def run(self):
        """Start the GUI."""
        self.root.mainloop()

if __name__ == "__main__":

```

```
csv_path = "../Kaitlin's Workspace/arasaac_synset_mapping_20251106_130530.csv"
```

```
board = AACBoard(csv_path)
board.run()
```

Implementation complete. Summary:

Summary

Integrated the image2text ↔ text2board workflow into the AAC board UI.

Features:

1. Dual input modes

- Text description: type a scenario description
- Image upload: upload an image (uses BLIP for captioning)

2. Workflow

- Image mode: Image ↔ BLIP caption ↔ Sentence transformer matching ↔ AAC board
- Text mode: Text ↔ Sentence transformer matching ↔ AAC board

3. Specificity slider

- : 10–30 items (default 20)
- Controls how many symbols appear on the board
- Lower = more focused (top 10)
- Higher = more options (top 30)

4. UI

- Simple Tkinter interface
- Radio buttons to switch between text/image input
- Slider with live value display
- Generate button to create the board
- Status label showing generation progress
- Scrollable board display with pictogram buttons

Technical details:

- Uses BLIP (Salesforce/blip-image-captioning-base) for image captioning
- Uses SentenceTransformer (all-MiniLM-L6-v2) for semantic text matching
- Pre-computes keyword embeddings at startup for faster generation
- Ranks keywords by cosine similarity and selects top N unique synsets
- Displays buttons with images and keywords from ARASAAC

The code is in a single file (aac_board.py) and keeps the original simple structure. Run it and it will:

1. Load the CSV and models (may take a moment on first run)
2. Allow you to enter text or upload an image
3. Adjust the specificity slider
4. Click "Generate Board" to create the AAC board

The board updates dynamically based on your input and specificity setting.

User

can you create a second version that mocks the version Kaitlin has in her workspace using Streamlit but using my entire workflow, so i want it to visually look the same but be Matt's ML algorithmns....

Cursor

Reviewing Kaitlin's Streamlit app to match its UI, then creating a Streamlit version using Matt's ML workflow.

Creating a Streamlit version matching Kaitlin's UI but using Matt's ML workflow:

Creating the Streamlit version:

```
import streamlit as st
from gtts import gTTS
import tempfile
from pathlib import Path
from PIL import Image
import pandas as pd
import torch
import numpy as np
from transformers import BlipProcessor, BlipForConditionalGeneration
from sentence_transformers import SentenceTransformer
import os

## Configuration ##
st.set_page_config(page_title="AAC Board Generator", layout="wide") # Widened layout
```

```

# Paths
CSV_PATH = "../Kaitlin's Workspace/arasaac_synset_mapping_20251106_130530.csv"

# Initialize session state
if 'result' not in st.session_state:
    st.session_state.result = None
if 'uploaded_image' not in st.session_state:
    st.session_state.uploaded_image = None

# Cache so it only loads once.
@st.cache_resource
def load_models():
    """Load BLIP and sentence transformer models."""
    device = "cuda" if torch.cuda.is_available() else "cpu"

    # Load BLIP for image captioning
    try:
        blip_processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
        blip_model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-
captioning-base").to(device)
        blip_loaded = True
    except Exception as e:
        st.error(f"Could not load BLIP model: {e}")
        blip_processor = None
        blip_model = None
        blip_loaded = False

    # Load sentence transformer
    try:
        sentence_model = SentenceTransformer("all-MiniLM-L6-v2")
        sentence_model_loaded = True
    except Exception as e:
        st.error(f"Could not load sentence transformer: {e}")
        sentence_model = None
        sentence_model_loaded = False

    return {
        'blip_processor': blip_processor,
        'blip_model': blip_model,
        'blip_loaded': blip_loaded,
        'sentence_model': sentence_model,
        'sentence_model_loaded': sentence_model_loaded,
        'device': device
    }

@st.cache_data
def load_csv_data():
    """Load and preprocess CSV data."""
    df = pd.read_csv(CSV_PATH)
    df = df.drop_duplicates(subset=['primary_keyword'])
    return df

```

```

@st.cache_data
def precompute_embeddings(df, sentence_model):
    """Pre-compute keyword embeddings."""
    keywords = df["primary_keyword"].tolist()
    embeddings = sentence_model.encode(keywords, show_progress_bar=False,
convert_to_numpy=True)
    df = df.copy()
    df["keyword_emb"] = [emb for emb in embeddings]
    return df


def caption_image(image, models):
    """Generate caption from image using BLIP."""
    if not models['blip_loaded']:
        raise Exception("BLIP model not loaded")

    image = image.convert("RGB")
    inputs = models['blip_processor'](images=image, return_tensors="pt").to(models['device'])

    with torch.no_grad():
        out = models['blip_model'].generate(**inputs, max_length=30, num_beams=5,
repetition_penalty=1.15)

    caption = models['blip_processor'].decode(out[0], skip_special_tokens=True).strip()
    return caption


def text_to_board(text, df_with_embeddings, sentence_model, top_n=20):
    """Convert text to ranked symbols using sentence transformers."""
    # Encode input text
    text_emb = sentence_model.encode(text, convert_to_numpy=True)

    # Calculate similarity scores (cosine similarity)
    df_with_embeddings = df_with_embeddings.copy()
    df_with_embeddings["score"] = df_with_embeddings["keyword_emb"].apply(
        lambda emb: np.dot(text_emb, emb) / (np.linalg.norm(text_emb) * np.linalg.norm(emb)))

    # Sort by score and get top N
    df_sorted = df_with_embeddings.sort_values("score", ascending=False)
    top_results = df_sorted.head(top_n * 2) # Get more to account for duplicates

    # Extract unique symbols (prefer highest scoring)
    symbols = []
    seen_synsets = set()
    for _, row in top_results.iterrows():
        synset = row['synset']
        if synset not in seen_synsets:
            symbols.append({
                'synset': synset,
                'pictogram_id': row['pictogram_id'],
                'primary_keyword': row['primary_keyword'],
                'image_url': row['image_url'],
                'keyword_list': [row['primary_keyword']], # Format for compatibility

```

```

        'score': row['score']
    })
    seen_synsets.add(synset)
    if len(symbols) >= top_n:
        break

return symbols

# Load models and data
models = load_models()
df = load_csv_data()

# Pre-compute embeddings if sentence model is loaded
if models['sentence_model_loaded']:
    df_with_embeddings = precompute_embeddings(df, models['sentence_model'])
else:
    df_with_embeddings = None

# App Title
st.title("Dynamic AAC Board Generator")
st.write("Generate communication boards from images or text descriptions")

# Sidebar for input selection AND image display
with st.sidebar:
    st.header("🔧 Controls")

    input_type = st.radio("Input Type:", ["Upload Image", "Text Description"])

    # Specificity slider (Matt's feature)
    specificity = st.slider("Specificity (number of symbols):", 10, 30, 20, 1)

    # Show uploaded image in sidebar if it exists.
    if st.session_state.uploaded_image:
        st.image(st.session_state.uploaded_image, caption="Current Image",
use_container_width=True)

    # Clear board button.
    if st.session_state.result:
        if st.button("🗑️ Clear Board"):
            st.session_state.result = None
            st.session_state.uploaded_image = None
            st.rerun()

# Main content area
if input_type == "Upload Image":
    uploaded_file = st.file_uploader("Choose an image", type=['jpg', 'jpeg', 'png'])

    if uploaded_file:
        # Store image in session state.
        image = Image.open(uploaded_file)
        st.session_state.uploaded_image = image

        if st.button("Generate Board", type="primary"):

```



```

if not models['blip_loaded']:
    st.error("BLIP model not available. Please install transformers.")
elif not models['sentence_model_loaded']:
    st.error("Sentence transformer model not available.")
else:
    with st.spinner("Generating AAC board..."):
        # Generate caption from image using BLIP
        caption = caption_image(image, models)
        st.info(f"Generated caption: {caption}")

        # Convert caption to board using sentence transformers
        board = text_to_board(caption, df_with_embeddings, models['sentence_model'],
top_n=specificity)

        st.session_state.result = {
            'captions': [caption],
            'combined_caption': caption,
            'board': board
        }
        st.success(f"Generated {len(board)} symbols!")

else: # Text Description
    text_input = st.text_area("Enter scenario description:",
        placeholder="e.g., Children playing at the playground",
        height=100)

    if st.button("Generate Board", type="primary") and text_input:
        if not models['sentence_model_loaded']:
            st.error("Sentence transformer model not available.")
        else:
            with st.spinner("Generating AAC board..."):
                # Use text-to-board pipeline with sentence transformers
                board = text_to_board(text_input, df_with_embeddings, models['sentence_model'],
top_n=specificity)

                st.session_state.result = {
                    'captions': [text_input],
                    'combined_caption': text_input,
                    'board': board
                }
                st.session_state.uploaded_image = None # Clear image when using text.
                st.success(f"Generated {len(board)} symbols!")

# Display Board - use session_state result.
if st.session_state.result:
    result = st.session_state.result

    st.markdown("---")
    st.subheader("Generated AAC Board")
    st.write(f"Scene: {result['combined_caption']}")

    # Show more symbols with responsive columns
    num_symbols = st.slider("Number of symbols to display:", 8, len(result['board']), min(24,
len(result['board'])), 4)

```

```

cols_per_row = st.slider("Symbols per row:", 4, 8, 6, 1)

board = result['board'][:num_symbols]
st.write(f"Displaying {len(board)} of {len(result['board'])} total symbols")

for row_start in range(0, len(board), cols_per_row):
    cols = st.columns(cols_per_row)

    for idx, col in enumerate(cols):
        symbol_idx = row_start + idx
        if symbol_idx < len(board):
            symbol = board[symbol_idx]

            with col:
                # Display image
                st.image(symbol['image_url'], use_container_width=True)

                # Keyword label
                keyword = symbol['keyword_list'][0] if symbol['keyword_list'] else
symbol.get('primary_keyword', 'N/A')
                st.markdown(f"***{keyword}***", unsafe_allow_html=True)

                # Audio button
                if st.button(f"🔊", key=f"audio_{symbol_idx}"):
                    try:
                        tts = gTTS(text=keyword, lang='en', slow=False)

                        with tempfile.NamedTemporaryFile(delete=False, suffix='.mp3') as fp:
                            tts.save(fp.name)
                            st.audio(fp.name, format='audio/mp3', autoplay=True)

                    except Exception as e:
                        st.warning(f"Could not generate audio for '{keyword}'")

            else:
                # Welcome message
                st.info("Upload an image or enter a text description to generate an AAC board")

```

Created a Streamlit version that matches Kaitlin's UI and uses Matt's ML workflow. Summary:

Features:

1. Same visual layout as Kaitlin's app

:

- Wide layout with sidebar controls
- Radio buttons for input type selection
- Image display in sidebar
- Clear board button
- Same board display format with images, keywords, and audio buttons

2. Matt's ML algorithms

:

- **BLIP** for image captioning (instead of SceneCaptioner)
- **Sentence Transformers** for text-to-board matching (instead of ConceptExtractor + SynsetMapper)
- Direct keyword matching using cosine similarity

3. Specificity slider

- Added in the sidebar (10-30 items, default 20)
- Controls how many symbols are generated

4. Workflow

- **Image mode:** Image → BLIP caption → Sentence transformer matching → Board
- **Text mode:** Text → Sentence transformer matching → Board

5. Data format

- Uses the CSV file (arasaac_synset_mapping_20251106_130530.csv)
- Converts results to the same format Kaitlin's app expects
- Compatible with the existing display code

The file is saved as demo_app_streamlit.py in Matt's Workspace. It should look and behave like Kaitlin's app but uses Matt's ML pipeline. Models are cached with @st.cache_resource and @st.cache_data for performance.

To run it:

```
streamlit run "Matts Workspace/demo_app_streamlit.py"
streamlit run "Matts Workspace/demo_app_streamlit.py"
```

User

when running in streamlit i go:

```
streamlit.runtime.caching.cache_errors.UnhashableParamError: Cannot hash argument 'sentence_model' (of type sentence_transformers.SentenceTransformer.SentenceTransformer) in 'precompute_embeddings'.
```

To address this, you can tell Streamlit not to hash this argument by adding a

leading underscore to the argument's name in the function signature:

```
@st.cache_data
```

```
def precompute_embeddings(_sentence_model, ...):
```

```
...
```

```
File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/Matts  
Workspace/demo_app_streamlit.py", line 136, in
```

```
df_with_embeddings = precompute_embeddings(df, models['sentence_model'])
```

```
File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/  
python3.13/site-packages/streamlit/runtime/caching/cache_utils.py", line 228, in call
```

```
return self._get_or_create_cached_value(args, kwargs, spinner_message)
```

```
~~~~~^
```

```
File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/  
python3.13/site-packages/streamlit/runtime/caching/cache_utils.py", line 243, in  
_get_or_create_cached_value
```

```
value_key = _make_value_key(
```

```
    cache_type=self._info.cache_type,
```

```
...<3 lines>...
```

```
    hash_funcs=self._info.hash_funcs,
```

```
)
```

```
File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/  
python3.13/site-packages/streamlit/runtime/caching/cache_utils.py", line 476, in  
_make_value_key
```

```
raise UnhashableParamError(cache_type, func, arg_name, arg_value, exc)
```

```
File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/  
python3.13/site-packages/streamlit/runtime/caching/cache_utils.py", line 468, in  
_make_value_key
```

```
update_hash(
```

```
~~~~~^
```

```
    arg_value,
```

```
    ^
```

```
...<3 lines>...
```

```
    hash_source=func,
```

AAAAAAAAAAAAAAAAAAAA

)

^

File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/python3.13/site-packages/streamlit/runtime/caching/hashing.py", line 169, in update_hash

ch.update(hasher, val)

~~~~~^

File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/python3.13/site-packages/streamlit/runtime/caching/hashing.py", line 345, in update

b = self.to\_bytes(obj)

File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/python3.13/site-packages/streamlit/runtime/caching/hashing.py", line 327, in to\_bytes

b = b"%s:%s" % (tname, self.\_to\_bytes(obj))

~~~~~^

File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/python3.13/site-packages/streamlit/runtime/caching/hashing.py", line 650, in _to_bytes

self.update(h, item)

~~~~~^

File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/python3.13/site-packages/streamlit/runtime/caching/hashing.py", line 345, in update

b = self.to\_bytes(obj)

File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/python3.13/site-packages/streamlit/runtime/caching/hashing.py", line 327, in to\_bytes

b = b"%s:%s" % (tname, self.\_to\_bytes(obj))

~~~~~^

File "/Users/myurk/Desktop/Dynamic-Augmentative-and-Alternative-Communication/.venv/lib/python3.13/site-packages/streamlit/runtime/caching/hashing.py", line 647, in _to_bytes

raise UnhashableTypeError() from ex

Fixing the caching issue by prefixing the unhashable parameter with an underscore so Streamlit doesn't hash it:

Fixing the caching issue by prefixing the unhashable parameter with an underscore:

```
import streamlit as st
from gtts import gTTS
import tempfile
from pathlib import Path
from PIL import Image
import pandas as pd
import torch
import numpy as np
from transformers import BlipProcessor, BlipForConditionalGeneration
from sentence_transformers import SentenceTransformer
import os

## Configuration ##
st.set_page_config(page_title="AAC Board Generator", layout="wide") # Widened layout

# Paths
CSV_PATH = "../Kaitlin's Workspace/arasaac_synset_mapping_20251106_130530.csv"

# Initialize session state
if 'result' not in st.session_state:
    st.session_state.result = None
if 'uploaded_image' not in st.session_state:
    st.session_state.uploaded_image = None

# Cache so it only loads once.
@st.cache_resource
def load_models():
    """Load BLIP and sentence transformer models."""
    device = "cuda" if torch.cuda.is_available() else "cpu"

    # Load BLIP for image captioning
    try:
        blip_processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
        blip_model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-
captioning-base").to(device)
        blip_loaded = True
    except Exception as e:
        st.error(f"Could not load BLIP model: {e}")
        blip_processor = None
        blip_model = None
        blip_loaded = False

    # Load sentence transformer
    try:
        sentence_model = SentenceTransformer("all-MiniLM-L6-v2")
```

```

    sentence_model_loaded = True
except Exception as e:
    st.error(f"Could not load sentence transformer: {e}")
    sentence_model = None
    sentence_model_loaded = False

return {
    'blip_processor': blip_processor,
    'blip_model': blip_model,
    'blip_loaded': blip_loaded,
    'sentence_model': sentence_model,
    'sentence_model_loaded': sentence_model_loaded,
    'device': device
}

```

```

@st.cache_data
def load_csv_data():
    """Load and preprocess CSV data."""
    df = pd.read_csv(CSV_PATH)
    df = df.drop_duplicates(subset=['primary_keyword'])
    return df

```

```

@st.cache_data
def precompute_embeddings(df, _sentence_model):
    """Pre-compute keyword embeddings."""
    keywords = df["primary_keyword"].tolist()
    embeddings = _sentence_model.encode(keywords, show_progress_bar=False,
convert_to_numpy=True)
    df = df.copy()
    df["keyword_emb"] = [emb for emb in embeddings]
    return df

```

```

def caption_image(image, models):
    """Generate caption from image using BLIP."""
    if not models['blip_loaded']:
        raise Exception("BLIP model not loaded")

    image = image.convert("RGB")
    inputs = models['blip_processor'](images=image, return_tensors="pt").to(models['device'])

    with torch.no_grad():
        out = models['blip_model'].generate(**inputs, max_length=30, num_beams=5,
repetition_penalty=1.15)

    caption = models['blip_processor'].decode(out[0], skip_special_tokens=True).strip()
    return caption

```

```

def text_to_board(text, df_with_embeddings, sentence_model, top_n=20):
    """Convert text to ranked symbols using sentence transformers."""
    # Encode input text

```

```

text_emb = sentence_model.encode(text, convert_to_numpy=True)

# Calculate similarity scores (cosine similarity)
df_with_embeddings = df_with_embeddings.copy()
df_with_embeddings["score"] = df_with_embeddings["keyword_emb"].apply(
    lambda emb: np.dot(text_emb, emb) / (np.linalg.norm(text_emb) * np.linalg.norm(emb)))

# Sort by score and get top N
df_sorted = df_with_embeddings.sort_values("score", ascending=False)
top_results = df_sorted.head(top_n * 2) # Get more to account for duplicates

# Extract unique symbols (prefer highest scoring)
symbols = []
seen_synsets = set()
for _, row in top_results.iterrows():
    synset = row['synset']
    if synset not in seen_synsets:
        symbols.append({
            'synset': synset,
            'pictogram_id': row['pictogram_id'],
            'primary_keyword': row['primary_keyword'],
            'image_url': row['image_url'],
            'keyword_list': [row['primary_keyword']], # Format for compatibility
            'score': row['score']
        })
    seen_synsets.add(synset)
    if len(symbols) >= top_n:
        break

return symbols

# Load models and data
models = load_models()
df = load_csv_data()

# Pre-compute embeddings if sentence model is loaded
if models['sentence_model_loaded']:
    df_with_embeddings = precompute_embeddings(df, models['sentence_model'])
else:
    df_with_embeddings = None

# App Title
st.title("Dynamic AAC Board Generator")
st.write("Generate communication boards from images or text descriptions")

# Sidebar for input selection AND image display
with st.sidebar:
    st.header("🔧 Controls")

    input_type = st.radio("Input Type:", ["Upload Image", "Text Description"])

    # Specificity slider (Matt's feature)
    specificity = st.slider("Specificity (number of symbols):", 10, 30, 20, 1)

```



```

# Show uploaded image in sidebar if it exists.
if st.session_state.uploaded_image:
    st.image(st.session_state.uploaded_image, caption="Current Image",
use_container_width=True)

# Clear board button.
if st.session_state.result:
    if st.button("✖ Clear Board"):
        st.session_state.result = None
        st.session_state.uploaded_image = None
        st.rerun()

# Main content area
if input_type == "Upload Image":
    uploaded_file = st.file_uploader("Choose an image", type=['jpg', 'jpeg', 'png'])

    if uploaded_file:
        # Store image in session state.
        image = Image.open(uploaded_file)
        st.session_state.uploaded_image = image

    if st.button("Generate Board", type="primary"):
        if not models['blip_loaded']:
            st.error("BLIP model not available. Please install transformers.")
        elif not models['sentence_model_loaded']:
            st.error("Sentence transformer model not available.")
        else:
            with st.spinner("Generating AAC board..."):
                # Generate caption from image using BLIP
                caption = caption_image(image, models)
                st.info(f"Generated caption: {caption}")

                # Convert caption to board using sentence transformers
                board = text_to_board(caption, df_with_embeddings, models['sentence_model'],
top_n=specificity)

                st.session_state.result = {
                    'captions': [caption],
                    'combined_caption': caption,
                    'board': board
                }
                st.success(f"Generated {len(board)} symbols!")

else: # Text Description
    text_input = st.text_area("Enter scenario description:",
        placeholder="e.g., Children playing at the playground",
        height=100)

    if st.button("Generate Board", type="primary") and text_input:
        if not models['sentence_model_loaded']:
            st.error("Sentence transformer model not available.")
        else:
            with st.spinner("Generating AAC board..."):

```

```

    # Use text-to-board pipeline with sentence transformers
    board = text_to_board(text_input, df_with_embeddings, models['sentence_model'],
top_n=specificity)

    st.session_state.result = {
        'captions': [text_input],
        'combined_caption': text_input,
        'board': board
    }
    st.session_state.uploaded_image = None # Clear image when using text.
    st.success(f"Generated {len(board)} symbols!")

# Display Board - use session_state result.
if st.session_state.result:
    result = st.session_state.result

    st.markdown("---")
    st.subheader("Generated AAC Board")
    st.write(f"**Scene:** {result['combined_caption']}")

    # Show more symbols with responsive columns
    num_symbols = st.slider("Number of symbols to display:", 8, len(result['board']), min(24,
len(result['board'])), 4)
    cols_per_row = st.slider("Symbols per row:", 4, 8, 6, 1)

    board = result['board'][:num_symbols]
    st.write(f"Displaying {len(board)} of {len(result['board'])} total symbols")

    for row_start in range(0, len(board), cols_per_row):
        cols = st.columns(cols_per_row)

        for idx, col in enumerate(cols):
            symbol_idx = row_start + idx
            if symbol_idx < len(board):
                symbol = board[symbol_idx]

            with col:
                # Display image
                st.image(symbol['image_url'], use_container_width=True)

                # Keyword label
                keyword = symbol['keyword_list'][0] if symbol['keyword_list'] else
symbol.get('primary_keyword', 'N/A')
                st.markdown(f"**{keyword}**", unsafe_allow_html=True)

            # Audio button
            if st.button(f"🔊", "key=f"audio_{symbol_idx}"):
                try:
                    tts = gTTS(text=keyword, lang='en', slow=False)

                    with tempfile.NamedTemporaryFile(delete=False, suffix='.mp3') as fp:
                        tts.save(fp.name)
                        st.audio(fp.name, format='audio/mp3', autoplay=True)

```

```
except Exception as e:  
    st.warning(f"Could not generate audio for '{keyword}'")
```

else:

```
# Welcome message  
st.info("Upload an image or enter a text description to generate an AAC board")
```

Fixed the caching issue. The problem was that Streamlit can't hash the `sentence_model` object for caching. I added an underscore prefix (`_sentence_model`) to the parameter name, which tells Streamlit not to hash it.

The function will still use the model to compute embeddings, but Streamlit will only cache based on the `df` parameter. Since the model is cached separately with `@st.cache_resource` and we always use the same instance, this works correctly.

The app should now run without the `UnhashableParamError`. Try running it again:

```
streamlit run "Matt's Workspace/demo_app_streamlit.py"  
streamlit run "Matt's Workspace/demo_app_streamlit.py"
```