```
# streamlit_app.py
# ------------------------------------------------------------
# Streamlit app: Upload teacher-guide PDFs, align to standards,
# generate citations with evaluation + validation workflow.
# ------------------------------------------------------------

import io
import json
import textwrap
from dataclasses import dataclass, asdict
from typing import List, Dict, Tuple

import numpy as np
import pandas as pd
import streamlit as st

# PDF parsing
try:
    import fitz  # PyMuPDF
except Exception as e:
    st.error("PyMuPDF (fitz) is required. Install with: pip install pymupdf")
    raise

# Embeddings
try:
    from sentence_transformers import SentenceTransformer
except Exception:
    st.error("sentence-transformers is required. Install with: pip install sentence-
transformers")
    raise

from sklearn.metrics.pairwise import cosine_similarity

# ---------------------------
# Config
# ---------------------------
st.set_page_config(page_title="Standards Alignment & Citation Generator",
layout="wide")

# ---------------------------
# Data classes
# ---------------------------
@dataclass
class PDFChunk:
```

```
    doc_name: str
    page: int
    chunk_id: str
    text: str

@dataclass
class Citation:
    standard_code: str
    standard_text: str
    doc_name: str
    page: int
    chunk_id: str
    snippet: str
    similarity: float

@dataclass
class EvalResult:
    standard_code: str
    coverage_score: float
    precision_score: float
    clarity_score: float
    overall_score: float
    details: Dict

# ---------------------------
# Utilities
# ---------------------------

def chunk_text(text: str, max_chars: int = 900, overlap: int = 150) -> List[str]:
    """Simple character-based chunker with overlap to preserve context."""
    text = text.replace("\u00ad", "")  # soft hyphens
    chunks = []
    start = 0
    while start < len(text):
        end = min(start + max_chars, len(text))
        chunk = text[start:end].strip()
        if chunk:
            chunks.append(chunk)
        start = end - overlap
        if start < 0:
            start = 0
        if end == len(text):
            break
    return chunks
```

```
def extract_pdf_chunks(file_bytes: bytes, doc_name: str, max_chars: int, overlap:
int) -> List[PDFChunk]:
    doc = fitz.open(stream=file_bytes, filetype="pdf")
    all_chunks: List[PDFChunk] = []
    for page_index in range(len(doc)):
        page = doc[page_index]
        text = page.get_text("text") or ""
        # Skip empty pages
        if not text.strip():
            continue
        chunks = chunk_text(text, max_chars=max_chars, overlap=overlap)
        for i, c in enumerate(chunks):
            all_chunks.append(PDFChunk(
                doc_name=doc_name,
                page=page_index + 1,
                chunk_id=f"{doc_name}-p{page_index+1}-c{i+1}",
                text=c,
            ))
    return all_chunks


@st.cache_resource(show_spinner=False)
def load_embedding_model(model_name: str = "sentence-transformers/all-
MiniLM-L6-v2"):
    return SentenceTransformer(model_name)


def embed_texts(model: SentenceTransformer, texts: List[str]) -> np.ndarray:
    emb = model.encode(texts, normalize_embeddings=True,
show_progress_bar=False)
    return np.array(emb)


def align_standard_to_chunks(std_text: str, chunk_texts: List[str], chunk_meta:
List[PDFChunk], model: SentenceTransformer, top_k: int = 5) -> List[Citation]:
    std_emb = embed_texts(model, [std_text])  # (1, d)
    ch_embs = embed_texts(model, chunk_texts)  # (n, d)
    sims = cosine_similarity(std_emb, ch_embs)[0]
    idx = np.argsort(-sims)[:top_k]
    citations: List[Citation] = []
    for i in idx:
        citations.append(Citation(
```

```
            standard_code="",
            standard_text=std_text,
            doc_name=chunk_meta[i].doc_name,
            page=chunk_meta[i].page,
            chunk_id=chunk_meta[i].chunk_id,
            snippet=chunk_meta[i].text[:650],
            similarity=float(sims[i]),
        ))
    return citations


def heuristic_evaluate_alignment(std_text: str, citations: List[Citation]) ->
EvalResult:
    # Working theory: good alignment has multiple high-sim hits spread across
docs/pages
    sims = np.array([c.similarity for c in citations]) if citations else np.array([0.0])
    # Coverage: proportion above 0.6 (tunable)
    coverage = float(np.mean(sims >= 0.60)) if len(sims) else 0.0
    # Precision: top-1 vs top-5 drop-off (sharper is better up to a point)
    if len(sims) >= 2:
        precision = float(max(0.0, min(1.0, (sims[0] - np.mean(sims[1:])) * 2)))
    else:
        precision = float(min(1.0, sims[0]))
    # Clarity: proxy using snippet readability (shorter chunks + presence of action
verbs)
    verbs = ["explain", "model", "justify", "argue", "construct", "represent", "solve",
"reason", "use", "attend"]
    verb_hits = 0
    total = 0
    for c in citations:
        total += 1
        snip_lower = c.snippet.lower()
        if any(v in snip_lower for v in verbs) and len(c.snippet) > 200:
            verb_hits += 1
    clarity = float(verb_hits / total) if total else 0.0
    # Overall = weighted blend (tunable)
    overall = round(float(0.4 * coverage + 0.35 * precision + 0.25 * clarity), 3)
    return EvalResult(
        standard_code="",
        coverage_score=round(coverage, 3),
        precision_score=round(precision, 3),
        clarity_score=round(clarity, 3),
        overall_score=overall,
        details={
```

```
        "similarities": [round(s, 4) for s in sims.tolist()],
        "thresholds": {"coverage_ge": 0.60},
    },
)


def make_download(data: pd.DataFrame, filename: str, label: str):
    csv_bytes = data.to_csv(index=False).encode("utf-8")
    st.download_button(label=label, data=csv_bytes, file_name=filename,
mime="text/csv")


# ---------------------------
# UI
# ---------------------------
st.title("📚 Standards Alignment & Citation Generator")
st.caption("Upload teacher-guide PDFs and a standards list. Get auto-suggested
citations, quick evals, and a validation workflow.")

with st.sidebar:
    st.header("Inputs")
    pdf_files = st.file_uploader("Upload Teacher Guide PDFs", type=["pdf"],
accept_multiple_files=True)
    st.markdown("**Standards CSV** (columns: `code`, `description`).")
    standards_file = st.file_uploader("Upload Standards CSV", type=["csv"])

    st.divider()
    st.subheader("Embedding & Chunking")
    model_name = st.text_input("Embedding model", value="sentence-
transformers/all-MiniLM-L6-v2")
    max_chars = st.slider("Chunk size (chars)", min_value=400, max_value=1600,
value=900, step=50)
    overlap = st.slider("Overlap (chars)", min_value=50, max_value=400,
value=150, step=10)
    top_k = st.slider("Top citations per standard", min_value=3, max_value=15,
value=5)

    st.divider()
    st.subheader("Filters")
    min_similarity = st.slider("Minimum similarity to surface", min_value=0.0,
max_value=1.0, value=0.55, step=0.01)


# Session state containers
```

```
if "pdf_chunks" not in st.session_state:
    st.session_state.pdf_chunks: List[PDFChunk] = []

if "standards_df" not in st.session_state:
    st.session_state.standards_df = pd.DataFrame(columns=["code",
"description"])

# Load & parse PDFs
with st.expander("1) Parse PDFs", expanded=True):
    if pdf_files:
        run_parse = st.button("Extract text from PDFs", type="primary")
        if run_parse:
            all_chunks: List[PDFChunk] = []
            for f in pdf_files:
                file_bytes = f.read()
                doc_name = f.name
                with st.spinner(f"Parsing {doc_name}..."):
                    chunks = extract_pdf_chunks(file_bytes, doc_name, max_chars,
overlap)
                    all_chunks.extend(chunks)
            st.session_state.pdf_chunks = all_chunks
            st.success(f"Parsed {len(pdf_files)} PDFs → {len(all_chunks)} chunks")
    else:
        st.info("Upload at least one PDF to continue.")

# Load standards
with st.expander("2) Load Standards", expanded=True):
    if standards_file is not None:
        try:
            df = pd.read_csv(standards_file).fillna("")
            # Normalize column names
            cols = {c.lower().strip(): c for c in df.columns}
            code_col = next((c for c in df.columns if c.lower().strip() == "code"), None)
            desc_col = next((c for c in df.columns if c.lower().strip() == "description"),
None)
            if code_col is None or desc_col is None:
                st.error("CSV must include 'code' and 'description' columns.")
            else:
                df = df.rename(columns={code_col: "code", desc_col: "description"})
                st.session_state.standards_df = df
                st.success(f"Loaded {len(df)} standards.")
                st.dataframe(df.head(10), use_container_width=True)
        except Exception as e:
            st.error(f"Failed to parse CSV: {e}")
```

```
    else:
        st.info("Upload a standards CSV.")

# Alignment & citations
with st.expander("3) Generate Citations & Eval", expanded=True):
    if st.session_state.pdf_chunks and not st.session_state.standards_df.empty:
        go = st.button("Run alignment", type="primary")
        if go:
            # Prepare model and data
            model = load_embedding_model(model_name)
            chunk_texts = [c.text for c in st.session_state.pdf_chunks]
            chunk_meta = st.session_state.pdf_chunks

            rows = []
            eval_rows = []

            for _, row in st.session_state.standards_df.iterrows():
                code = str(row["code"]).strip()
                desc = str(row["description"]).strip()
                citations = align_standard_to_chunks(desc, chunk_texts, chunk_meta,
model=model, top_k=top_k)
                # Attach code to each citation and filter by similarity
                citations = [c for c in citations if c.similarity >= min_similarity]
                for c in citations:
                    c.standard_code = code
                # Evaluation
                eval_res = heuristic_evaluate_alignment(desc, citations)
                eval_res.standard_code = code

                for c in citations:
                    rows.append({
                        "standard_code": code,
                        "standard_text": desc,
                        "doc_name": c.doc_name,
                        "page": c.page,
                        "chunk_id": c.chunk_id,
                        "similarity": round(c.similarity, 4),
                        "snippet": c.snippet,
                    })

                eval_rows.append({
                    "standard_code": code,
                    "coverage_score": eval_res.coverage_score,
                    "precision_score": eval_res.precision_score,
```

```
                "clarity_score": eval_res.clarity_score,
                "overall_score": eval_res.overall_score,
                "similarities": json.dumps(eval_res.details["similarities"]),
            })

        cites_df = pd.DataFrame(rows)
        eval_df = pd.DataFrame(eval_rows)
        if cites_df.empty:
            st.warning("No citations met the similarity threshold. Try lowering the
threshold or increasing top_k.")
        else:
            st.success(f"Generated {len(cites_df)} citations across {len(eval_df)}
standards.")
            st.dataframe(cites_df, use_container_width=True)
            make_download(cites_df, "citations.csv", "⬇️ Download citations.csv")

        st.subheader("Evaluation Summary")
        if not eval_df.empty:
            st.dataframe(eval_df.sort_values("overall_score", ascending=False),
use_container_width=True)
            make_download(eval_df, "evaluation.csv", "⬇️ Download
evaluation.csv")
        st.session_state.citations_df = cites_df
        st.session_state.eval_df = eval_df
    else:
        st.info("Complete steps 1 and 2, then click Run alignment.")

# Validation workflow
with st.expander("4) Human Validation", expanded=False):
    if "citations_df" in st.session_state and not st.session_state.citations_df.empty:
        cites_df = st.session_state.citations_df.copy()
        # Add columns for validator input if not present
        for col in ["valid?", "validator_notes"]:
            if col not in cites_df.columns:
                cites_df[col] = ""
        st.markdown("Mark each citation as valid/invalid and add notes. Then
export.")
        edited = st.data_editor(cites_df, num_rows="dynamic",
use_container_width=True, key="validator_table")
        make_download(edited, "validated_citations.csv", "⬇️ Download
validated_citations.csv")
    else:
        st.info("Run alignment to populate citations for validation.")
```

```
# Help & template
with st.expander("Help & Templates", expanded=False):
    st.markdown(
        """
        **Standards CSV template**

        ```csv
        code,description
        MP1,Make sense of problems and persevere in solving them.
        MP2,Reason abstractly and quantitatively.
        MP3,Construct viable arguments and critique the reasoning of others.
        MP4,Model with mathematics.
        MP5,Use appropriate tools strategically.
        MP6,Attend to precision.
        MP7,Look for and make use of structure.
        MP8,Look for and express regularity in repeated reasoning.
        ```

        Tips
        - Similarity threshold ~0.55–0.65 is a useful starting band.
        - Increase chunk size for more context; increase overlap to avoid cutting sentences.
        - The evaluation is heuristic. Treat it as a *working theory* and pair with human validation.
        - For speed on large documents, try a smaller embedding model or limit to specific units/pages.
        - To export a polished report, download the CSVs and format externally, or adapt this app to create PDF/Docs.
        """
)

# Footer
st.caption("MVP · Local embeddings (no API key). Replace the model or add RAG if you want more power.")
```