```
# Dr. Stephen Dietrich-Kolokouris -- Portfolio RAG Interface
# Complete production build: premium UI + full backend.

import hashlib
import json
import logging
import os
import re
import tempfile
from io import BytesIO
from typing import Dict, List, Optional, Tuple

import streamlit as st
from langchain_community.document_loaders import PyPDFDirectoryLoader
from langchain_community.vectorstores import FAISS
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings, ChatOpenAI

try:
    from reportlab.lib.pagesizes import LETTER
    from reportlab.pdfgen import canvas
    from reportlab.lib.units import inch
    REPORTLAB_OK = True
except Exception:
    REPORTLAB_OK = False

try:
    from scoring import score_overall
    from mitigations import tier_from_score, mitigation_playbook
    SCORING_ENABLED = True
except Exception:
    SCORING_ENABLED = False

logger = logging.getLogger(__name__)


# ═══════════════════════════════════════════════════════════════════════════
# PAGE CONFIG
# ═══════════════════════════════════════════════════════════════════════════
st.set_page_config(
    page_title="Dr. Stephen Dietrich-Kolokouris",
    page_icon="◆",
    layout="wide",
    initial_sidebar_state="expanded",
```

```
)


# ============================================================================
# GUARDRAILS
# ============================================================================
_EXTERNAL_REF_REGEX = re.compile(
    r"(^\s*works\s+cited\b|^\s*references\s*$|^\s*bibliography\b|https?://|www\.)
    flags=re.IGNORECASE | re.MULTILINE,
)

def enforce_no_external_refs(text: str) -> str:
    if not text:
        return text
    if _EXTERNAL_REF_REGEX.search(text):
        return (
            "Response blocked: external citation or link pattern detected. "
            "This system can only cite the loaded corpus."
        )
    return text


# ============================================================================
# ASSET HELPERS
# ============================================================================
def safe_exists(path: Optional[str]) -> bool:
    try:
        return bool(path) and os.path.exists(path)
    except Exception:
        return False

def first_existing(paths: List[str]) -> Optional[str]:
    for p in paths:
        if safe_exists(p):
            return p
    return None

def _read_file_safe(path: str) -> str:
    try:
        with open(path, "r", encoding="utf-8") as f:
            return f.read()
    except Exception:
        return ""

LOGO_PATH = first_existing([
    os.path.join("assets", "logo.png"), os.path.join("assets", "logo.jpg"),
    "logo.png", "logo.jpg",
```

```
])
HEADSHOT_PATH = first_existing([
    os.path.join("assets", "headshot.png"), os.path.join("assets", "headshot.jpg"
    os.path.join("assets", "headshot.jpeg"),
    "headshot.png", "headshot.jpg", "headshot.jpeg",
])
ABOUT_MD_PATH = first_existing([os.path.join("assets", "about_stephen.md"), "abou
THINK_MD_PATH = first_existing([os.path.join("assets", "how_i_think.md"), "how_i_
PUBS_CSV_PATH = first_existing([os.path.join("assets", "publications.csv"), "publ
LINKEDIN_URL = "https://www.linkedin.com/in/stephendietrich-kolokouris/"


# ============================================================================
# LLM + EMBEDDINGS
# ============================================================================
def init_llm() -> ChatOpenAI:
    try:
        return ChatOpenAI(model="gpt-4o", temperature=0, api_key=st.secrets["OPEN
    except TypeError:
        return ChatOpenAI(model="gpt-4o", temperature=0, openai_api_key=st.secret

def init_embeddings() -> OpenAIEmbeddings:
    try:
        return OpenAIEmbeddings(api_key=st.secrets["OPENAI_API_KEY"])
    except TypeError:
        return OpenAIEmbeddings(openai_api_key=st.secrets["OPENAI_API_KEY"])


# ============================================================================
# FAISS PERSISTENCE + SHA-256 MANIFEST
# ============================================================================
FAISS_DIR = "faiss_index"
MANIFEST_PATH = os.path.join(FAISS_DIR, "manifest.json")

def _file_sha256(path: str) -> str:
    h = hashlib.sha256()
    with open(path, "rb") as f:
        for chunk in iter(lambda: f.read(1024 * 1024), b""):
            h.update(chunk)
    return h.hexdigest()

def _build_manifest(data_dir: str) -> dict:
    files = []
    for root, _, fnames in os.walk(data_dir):
        for fn in fnames:
            if fn.lower().endswith(".pdf"):
                p = os.path.join(root, fn)
```

```
                try:
                    files.append({
                        "path": p.replace("\\", "/"),
                        "sha256": _file_sha256(p),
                        "size": os.path.getsize(p),
                    })
                except Exception:
                    continue
    files = sorted(files, key=lambda x: x["path"])
    return {"data_dir": data_dir.replace("\\", "/"), "files": files}

def _manifest_changed(new_manifest: dict) -> bool:
    if not os.path.exists(MANIFEST_PATH):
        return True
    try:
        with open(MANIFEST_PATH, "r", encoding="utf-8") as f:
            old = json.loads(f.read())
        return old != new_manifest
    except Exception:
        return True

def load_or_build_faiss() -> FAISS:
    if not os.path.exists("data"):
        st.error("Missing **/data** directory. Commit/upload your PDFs into `/dat
        st.stop()

    embeddings = init_embeddings()
    new_manifest = _build_manifest("data")

    if os.path.isdir(FAISS_DIR) and not _manifest_changed(new_manifest):
        try:
            return FAISS.load_local(FAISS_DIR, embeddings, allow_dangerous_deseri
        except Exception as e:
            st.warning(f"FAISS index load failed; rebuilding. Reason: {e}")

    loader = PyPDFDirectoryLoader("data/")
    docs = loader.load()
    if not docs:
        st.error("No documents found in `/data`.")
        st.stop()

    splitter = RecursiveCharacterTextSplitter(chunk_size=1100, chunk_overlap=160)
    chunks = splitter.split_documents(docs)

    with st.status("Indexing corpus…", expanded=False) as status:
        vs = FAISS.from_documents(chunks, embeddings)
        status.update(label="Corpus indexed", state="complete")
```

```
    os.makedirs(FAISS_DIR, exist_ok=True)
    try:
        vs.save_local(FAISS_DIR)
        with open(MANIFEST_PATH, "w", encoding="utf-8") as f:
            json.dump(new_manifest, f, indent=2)
    except Exception as e:
        st.warning(f"FAISS index could not be saved (non-fatal): {e}")

    return vs


@st.cache_resource
def init_retriever():
    vs = load_or_build_faiss()
    return vs.as_retriever(
        search_type="mmr",
        search_kwargs={"k": 7, "fetch_k": 24, "lambda_mult": 0.5},
    )

retriever = init_retriever()


# ============================================================================
# EVIDENCE PACK + CITATIONS
# ============================================================================
def _page_label(meta: dict) -> Optional[str]:
    for key in ("page", "page_number", "loc.page_number"):
        if key in meta:
            try:
                p = int(meta[key])
                return f"p.{p+1}"
            except Exception:
                pass
    return None

def format_evidence_pack(docs) -> Tuple[str, List[str], List[str]]:
    seen = set()
    labels: List[str] = []
    files_only: List[str] = []
    parts: List[str] = []

    for d in docs:
        src_full = d.metadata.get("source", "") or ""
        src = os.path.basename(src_full) if src_full else "unknown"
        page = _page_label(d.metadata or {})
        label = f"{src} ({page})" if page else src
```

```
        uniq = (src, page, (d.page_content or "")[:80])
        if uniq in seen:
            continue
        seen.add(uniq)

        text = (d.page_content or "").strip()
        if not text:
            continue
        if len(text) > 2200:
            text = text[:2200].rstrip() + "…"

        parts.append(f"[SOURCE: {label}]\n{text}")
        labels.append(label)
        files_only.append(src)

    return "\n\n".join(parts), labels, sorted(set(files_only))


# ═══════════════════════════════════════════════════════════════════════════
# RECRUITER CONTEXT EXTRACTION
# ═══════════════════════════════════════════════════════════════════════════
_EMPTY_RECRUITER_STATE = {
    "target_roles": [], "domains": [], "location": None,
    "onsite_remote": None, "must_haves": [], "nice_to_haves": [],
    "dealbreakers": [],
}

def _dedupe_keep_order(items: List[str]) -> List[str]:
    seen = set()
    out: List[str] = []
    for x in items or []:
        x = (x or "").strip()
        if x and x.lower() not in seen:
            seen.add(x.lower())
            out.append(x)
    return out

def extract_recruiter_constraints(llm: ChatOpenAI, user_message: str) -> dict:
    prompt = (
        "Extract recruiter constraints from the message if present.\n"
        "Return JSON only matching this schema:\n"
        f"{json.dumps(_EMPTY_RECRUITER_STATE)}\n\n"
        "If not mentioned, use empty lists or null.\n"
        "onsite_remote: onsite | hybrid | remote | null.\n"
        "Keep items short.\n\n"
        f"MESSAGE:\n{user_message}\n\nJSON:"
```

```
    )
    try:
        out = llm.invoke(prompt)
        text = (getattr(out, "content", None) or str(out)).strip()
        text = re.sub(r"^```(?:json)?\s*", "", text)
        text = re.sub(r"\s*```$", "", text)
        data = json.loads(text)
        return data if isinstance(data, dict) else {}
    except Exception as e:
        logger.warning("Constraint extraction failed: %s", e)
        return {}

def update_recruiter_state(new_bits: dict):
    if not new_bits:
        return
    s = st.session_state.recruiter_state
    for k in ("target_roles", "domains", "must_haves", "nice_to_haves", "dealbrea
        if k in new_bits and isinstance(new_bits[k], list):
            s[k] = _dedupe_keep_order((s.get(k) or []) + new_bits[k])
    loc = new_bits.get("location")
    if isinstance(loc, str) and loc.strip():
        s["location"] = loc.strip()
    o = new_bits.get("onsite_remote")
    if o in ("onsite", "hybrid", "remote"):
        s["onsite_remote"] = o


# ═══════════════════════════════════════════════════════════════════════════
# QUERY REWRITE (history + recruiter context aware)
# ═══════════════════════════════════════════════════════════════════════════
def rewrite_to_standalone(
    llm: ChatOpenAI, chat_history: List[Dict],
    user_input: str, recruiter_state: dict, max_turns: int = 8,
) -> str:
    hist_lines: List[str] = []
    for m in chat_history[-max_turns:]:
        r = (m.get("role") or "").lower()
        if r in ("user", "assistant"):
            content = (m.get("content") or "").strip()
            if content:
                hist_lines.append(f"{r.upper()}: {content}")

    history_text = "\n".join(hist_lines).strip()
    state_text = json.dumps(recruiter_state or {}, ensure_ascii=False)

    prompt = (
        "Rewrite the user's last message into a standalone search query about "
```

```
        "Dr. Stephen Dietrich-Kolokouris. Resolve pronouns using history and cont
        "Don't add facts.\n\n"
        f"RECRUITER CONTEXT: {state_text}\n\n"
        f"CONVERSATION:\n{history_text}\n\n"
        f"USER MESSAGE:\n{user_input}\n\n"
        "STANDALONE QUERY:"
    )

    try:
        out = llm.invoke(prompt)
        text = (getattr(out, "content", None) or str(out)).strip()
        return text if text else user_input
    except Exception as e:
        logger.warning("Query rewrite failed (falling back to raw input): %s", e)
        return user_input


# ═══════════════════════════════════════════════════════════════════════════
# SYSTEM PROMPT BUILDER
# ═══════════════════════════════════════════════════════════════════════════
def build_system_prompt(
    personal_mode: bool,
    recruiter_state: dict,
    evidence_pack: str,
    vendor_block: str,
    action_mode: str = "chat",
) -> str:
    state_text = json.dumps(recruiter_state or {}, ensure_ascii=False)

    if personal_mode:
        tone_line = (
            "TONE MODE: Conversational.\n"
            "- You may include brief career context and lessons learned ONLY if s
            "- Keep it recruiter-friendly and precise. No hype.\n"
            "- Don't use phrases like 'based on the evidence pack' or 'according
            "just state the information naturally.\n"
        )
    else:
        tone_line = (
            "TONE MODE: Technical-only.\n"
            "- Direct, systems-focused, implementation-oriented.\n"
            "- Don't reference the evidence system — just state information natur
        )

    action_map = {
        "verify": (
            "TASK MODE: VERIFY.\n"
```

```
            "- Cross-check claims in the previous answer against the evidence.\n"
            "- Flag any claim not directly supported. Be honest.\n"
        ),
        "fit": (
            "TASK MODE: FIT SUMMARY.\n"
            "- Using the recruiter context JSON, produce a structured fit summary
            "  Strengths, Gaps or unknowns, Suggested next questions.\n"
            "- Use only evidence-backed claims.\n"
        ),
        "outreach": (
            "TASK MODE: OUTREACH.\n"
            "- Draft a recruiter outreach message (100–160 words) based on recrui
            "- Use only evidence-backed claims.\n"
            "- If key context is missing (role/location), ask ONE short question
        ),
    }
    action_instructions = action_map.get(action_mode, (
        "TASK MODE: CHAT.\n"
        "- Answer the question in a recruiter-grade, professional manner.\n"
        "- If something is not supported, say it's not in the current documentati
    ))

    return (
        "You are a professional assistant representing Dr. Stephen Dietrich-Kolok
        "MANDATORY CONSTRAINTS:\n"
        "1) Use ONLY the EVIDENCE PACK below (and vendor block if present).\n"
        "2) Do NOT invent facts, dates, employers, credentials, or project detail
        "3) If the answer cannot be supported, say it's not in the current docume
        "4) Do NOT include URLs or bibliography headings.\n"
        "5) Never reference 'the corpus', 'evidence pack', or 'the system' — spea
        f"RECRUITER CONTEXT JSON:\n{state_text}\n\n"
        + tone_line + "\n"
        + action_instructions + "\n\n"
        "EVIDENCE PACK:\n"
        + evidence_pack
        + vendor_block
    )


def build_vendor_block(vendor_ctx: Optional[dict]) -> str:
    if not vendor_ctx:
        return ""
    return (
        "\n\nSelected Vendor Context (deterministic):\n"
        f"- Vendor: {vendor_ctx.get('vendor_name')}\n"
        f"- Component: {vendor_ctx.get('product_or_component')}\n"
        f"- Class: {vendor_ctx.get('component_class')}\n"
```

```
        f"- Origin/Jurisdiction: {vendor_ctx.get('origin_jurisdiction')}\n"
        f"- Criticality: {vendor_ctx.get('criticality')}\n"
        f"- Tier: {vendor_ctx.get('tier')}\n"
        f"- Scores: REE={vendor_ctx.get('ree_risk')}, FW={vendor_ctx.get('firmwar
        f"Overall={vendor_ctx.get('overall_risk')}\n"
        "Mitigation priorities (deterministic):\n"
        + "\n".join(f"- {m}" for m in vendor_ctx.get("mitigations", []))
    )


# ═══════════════════════════════════════════════════════════════════════════
# PDF EXPORT
# ═══════════════════════════════════════════════════════════════════════════
def _sanitize_for_reportlab(text: str) -> str:
    if not text:
        return ""
    return re.sub(r"[\x00-\x08\x0b\x0c\x0e-\x1f]", "", text)

def _wrap_text_lines(text: str, max_chars: int = 95) -> List[str]:
    lines: List[str] = []
    for paragraph in text.split("\n"):
        paragraph = paragraph.strip()
        if not paragraph:
            lines.append("")
            continue
        while len(paragraph) > max_chars:
            split_at = paragraph.rfind(" ", 0, max_chars)
            if split_at <= 0:
                split_at = max_chars
            lines.append(paragraph[:split_at])
            paragraph = paragraph[split_at:].strip()
        lines.append(paragraph)
    return lines

def build_qa_pdf_bytes(messages: List[Dict], evidence_files: List[str]) -> Option
    if not REPORTLAB_OK:
        return None
    buf = BytesIO()
    c = canvas.Canvas(buf, pagesize=LETTER)
    w, h = LETTER
    margin = 0.75 * inch
    y = h - margin

    c.setFont("Helvetica-Bold", 14)
    c.drawString(margin, y, "Q&A Transcript — Dr. Stephen Dietrich-Kolokouris")
    y -= 24
    c.setFont("Helvetica", 9)
```

```
    c.drawString(margin, y, f"Evidence files: {', '.join(evidence_files) if evide
    y -= 20
    c.line(margin, y, w - margin, y)
    y -= 16

    for m in messages:
        role = (m.get("role") or "").upper()
        content = _sanitize_for_reportlab(m.get("content") or "")
        if not content:
            continue

        c.setFont("Helvetica-Bold", 10)
        if y < margin + 40:
            c.showPage()
            y = h - margin
        c.drawString(margin, y, f"{role}:")
        y -= 14

        c.setFont("Helvetica", 9)
        for line in _wrap_text_lines(content, 95):
            if y < margin + 20:
                c.showPage()
                y = h - margin
                c.setFont("Helvetica", 9)
            c.drawString(margin + 10, y, line)
            y -= 12
        y -= 8

    c.save()
    return buf.getvalue()


# ============================================================================
# CORE: run_turn()
# ============================================================================
def run_turn(user_text: str, action_mode: str = "chat") -> str:
    llm = init_llm()

    new_bits = extract_recruiter_constraints(llm, user_text)
    update_recruiter_state(new_bits)

    standalone_query = rewrite_to_standalone(
        llm, st.session_state.messages, user_text,
        st.session_state.recruiter_state, max_turns=8,
    )

    docs = retriever.invoke(standalone_query)
```

```
    if not docs:
        st.warning("Retrieval returned zero chunks for this query.")

    evidence_pack, evidence_labels, evidence_files = format_evidence_pack(docs)

    if evidence_files:
        existing = set(st.session_state.get("qa_evidence_files", []) or [])
        st.session_state.qa_evidence_files = sorted(existing.union(set(evidence_f

    vendor_block = build_vendor_block(st.session_state.get("selected_vendor_conte
    system_prompt = build_system_prompt(
        personal_mode=st.session_state.personal_mode,
        recruiter_state=st.session_state.recruiter_state,
        evidence_pack=evidence_pack,
        vendor_block=vendor_block,
        action_mode=action_mode,
    )

    try:
        out = llm.invoke([
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": user_text},
        ])
        answer = (out.content or "").strip()
    except Exception as e:
        answer = f"Sorry, I hit an error processing that. ({e})"

    answer = enforce_no_external_refs(answer)
    return answer


# ═══════════════════════════════════════════════════════════════════════════
# SESSION STATE
# ═══════════════════════════════════════════════════════════════════════════
if "messages" not in st.session_state:
    st.session_state.messages = []
if "pinned_opening" not in st.session_state:
    st.session_state.pinned_opening = True
if "recruiter_state" not in st.session_state:
    st.session_state.recruiter_state = dict(_EMPTY_RECRUITER_STATE)
if "personal_mode" not in st.session_state:
    st.session_state.personal_mode = True
if "qa_evidence_files" not in st.session_state:
    st.session_state.qa_evidence_files = []
```

```
# ═══════════════════════════════════════════════════════════════════════
# PREMIUM CSS
# ═══════════════════════════════════════════════════════════════════════
st.markdown("""
<style>
@import url('https://fonts.googleapis.com/css2?family=DM+Serif+Display&family=DM+

:root {
    --ink: #0d0f12;
    --surface: #f8f7f4;
    --surface-warm: #f2f0eb;
    --accent: #1a5c3a;
    --accent-light: #e8f0ec;
    --accent-glow: #2d8a5e;
    --text-primary: #1a1d23;
    --text-secondary: #5a5f6b;
    --text-muted: #8b909e;
    --border: #e2e0db;
    --border-light: #eceae5;
    --card-bg: #ffffff;
    --gold: #c9a84c;
    --shadow-sm: 0 1px 3px rgba(0,0,0,0.04);
    --shadow-md: 0 4px 16px rgba(0,0,0,0.06);
    --radius: 12px;
    --radius-sm: 8px;
    --transition: 0.25s cubic-bezier(0.4, 0, 0.2, 1);
}

html, body, [data-testid="stAppViewContainer"] {
    background-color: var(--surface) !important;
    font-family: 'DM Sans', -apple-system, sans-serif !important;
    color: var(--text-primary);
}
.stApp { background-color: var(--surface) !important; }

/* ── Sidebar ── */
[data-testid="stSidebar"] {
    background: linear-gradient(180deg, #0d0f12 0%, #141820 100%) !important;
    border-right: 1px solid rgba(255,255,255,0.06) !important;
}
[data-testid="stSidebar"] * { color: #c8cad0 !important; }
[data-testid="stSidebar"] .stMarkdown h1,
[data-testid="stSidebar"] .stMarkdown h2,
[data-testid="stSidebar"] .stMarkdown h3 {
    color: #ffffff !important;
    font-family: 'DM Serif Display', Georgia, serif !important;
}
```

```
[data-testid="stSidebar"] hr {
    border-color: rgba(255,255,255,0.08) !important;
    margin: 1.2rem 0 !important;
}

.sidebar-photo {
    width: 140px; height: 140px; border-radius: 50%;
    margin: 2rem auto 1rem; display: block;
    border: 3px solid rgba(74,222,128,0.3);
    box-shadow: 0 0 24px rgba(74,222,128,0.1);
    object-fit: cover;
}

.sidebar-photo-placeholder {
    width: 140px; height: 140px; border-radius: 50%;
    margin: 2rem auto 1rem;
    display: flex; align-items: center; justify-content: center;
    border: 3px solid rgba(74,222,128,0.3);
    box-shadow: 0 0 24px rgba(74,222,128,0.1);
    background: linear-gradient(135deg, #1a2332, #0d1520);
    font-size: 2.4rem; font-family: 'DM Serif Display', serif;
    color: #4ade80; letter-spacing: -1px;
}

.sidebar-name {
    text-align: center;
    font-family: 'DM Serif Display', Georgia, serif !important;
    font-size: 1.35rem; color: #ffffff !important;
    margin-bottom: 0.15rem; letter-spacing: -0.3px;
}

.sidebar-title-line {
    text-align: center; font-size: 0.78rem;
    color: #8b909e !important; letter-spacing: 0.5px;
    text-transform: uppercase; margin-bottom: 1.5rem;
}

.cred-row {
    display: flex; flex-wrap: wrap; gap: 6px;
    justify-content: center; margin-bottom: 1.2rem; padding: 0 0.5rem;
}
.cred-tag {
    display: inline-block; padding: 3px 10px; border-radius: 100px;
    font-size: 0.68rem; font-weight: 500; letter-spacing: 0.4px;
    font-family: 'JetBrains Mono', monospace;
}
.cred-tag.green { background: rgba(74,222,128,0.12); color: #4ade80; border: 1px
```

```
.cred-tag.gold  { background: rgba(201,168,76,0.12); color: #c9a84c; border: 1px
.cred-tag.blue  { background: rgba(96,165,250,0.12); color: #60a5fa; border: 1px

.sb-section-title {
    font-size: 0.7rem !important; font-weight: 500 !important;
    letter-spacing: 1.2px !important; text-transform: uppercase !important;
    color: #5a5f6b !important; margin-bottom: 0.6rem !important; padding-left: 2p
}
.sb-item { font-size: 0.85rem; color: #c8cad0; padding: 5px 0; line-height: 1.5;
.sb-item strong { color: #e2e4e8 !important; }

.sb-link {
    display: inline-flex; align-items: center; gap: 6px;
    padding: 8px 16px; border-radius: 8px;
    background: rgba(74,222,128,0.08); border: 1px solid rgba(74,222,128,0.15);
    color: #4ade80 !important; text-decoration: none !important;
    font-size: 0.82rem; font-weight: 500; transition: var(--transition); margin:
}
.sb-link:hover { background: rgba(74,222,128,0.15); border-color: rgba(74,222,128

/* ── Main ── */
.main-header { max-width: 960px; margin: 0 auto; padding: 2.5rem 0 1rem; }
.main-greeting {
    font-family: 'DM Serif Display', Georgia, serif;
    font-size: 2.2rem; color: var(--text-primary);
    letter-spacing: -0.8px; line-height: 1.15; margin-bottom: 0.3rem;
}
.main-subtitle {
    font-size: 1.05rem; color: var(--text-secondary);
    line-height: 1.6; max-width: 640px; margin-bottom: 2rem;
}

.domain-grid {
    display: grid; grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
    gap: 14px; margin-bottom: 2.5rem; max-width: 960px; margin-left: auto; margin
}
.domain-card {
    background: var(--card-bg); border: 1px solid var(--border);
    border-radius: var(--radius); padding: 1.25rem 1.3rem;
    transition: var(--transition); position: relative; overflow: hidden;
}
.domain-card:hover {
    border-color: var(--accent); box-shadow: var(--shadow-md); transform: transla
}
.domain-card::before {
    content: ''; position: absolute; top: 0; left: 0; right: 0; height: 3px;
    border-radius: var(--radius) var(--radius) 0 0;
```

```
}
.domain-card.cyber::before    { background: linear-gradient(90deg, #1a5c3a, #2d8a
.domain-card.rag::before      { background: linear-gradient(90deg, #3b82f6, #60a5
.domain-card.intel::before    { background: linear-gradient(90deg, #c9a84c, #dbbf
.domain-card.research::before { background: linear-gradient(90deg, #8b5cf6, #a78b

.domain-icon { font-size: 1.5rem; margin-bottom: 0.6rem; display: block; }
.domain-label { font-size: 0.92rem; font-weight: 600; color: var(--text-primary);
.domain-desc { font-size: 0.8rem; color: var(--text-muted); line-height: 1.45; }

.chat-section-label {
    font-size: 0.7rem; font-weight: 500; letter-spacing: 1.2px;
    text-transform: uppercase; color: var(--text-muted);
    margin-bottom: 0.8rem; padding-left: 2px;
    max-width: 960px; margin-left: auto; margin-right: auto;
}

/* ── Chat Messages ── */
[data-testid="stChatMessage"] {
    max-width: 960px !important; margin-left: auto !important; margin-right: auto
    border: none !important; background: transparent !important; padding: 0.8rem
}
[data-testid="stChatMessage"] p {
    font-family: 'DM Sans', sans-serif !important;
    font-size: 0.95rem !important; line-height: 1.7 !important;
    color: var(--text-primary) !important;
}

/* ── Chat Input ── */
[data-testid="stChatInput"] {
    max-width: 960px !important; margin-left: auto !important; margin-right: auto
}
[data-testid="stChatInput"] textarea {
    font-family: 'DM Sans', sans-serif !important; font-size: 0.95rem !important;
    border-radius: var(--radius) !important; border: 1px solid var(--border) !imp
    background: var(--card-bg) !important; padding: 1rem 1.2rem !important;
    transition: var(--transition);
}
[data-testid="stChatInput"] textarea:focus {
    border-color: var(--accent) !important; box-shadow: 0 0 0 3px var(--accent-li
}

/* ── Buttons ── */
.stButton > button {
    font-family: 'DM Sans', sans-serif !important; font-size: 0.82rem !important;
    font-weight: 500 !important; border-radius: var(--radius-sm) !important;
    border: 1px solid var(--border) !important; background: var(--card-bg) !impor
```

```
    color: var(--text-secondary) !important; padding: 0.5rem 1rem !important;
    transition: var(--transition);
}
.stButton > button:hover {
    border-color: var(--accent) !important; color: var(--accent) !important;
    background: var(--accent-light) !important;
}

.stDownloadButton > button {
    font-family: 'DM Sans', sans-serif !important; font-size: 0.82rem !important;
    border-radius: var(--radius-sm) !important;
    border: 1px solid var(--border) !important; background: var(--card-bg) !impor
    color: var(--text-secondary) !important;
}

/* ── Expanders ── */
[data-testid="stExpander"] {
    border: 1px solid rgba(255,255,255,0.06) !important;
    border-radius: 8px !important; background: rgba(255,255,255,0.02) !important;
}

/* ── Hide chrome ── */
#MainMenu, footer, header { visibility: hidden; }
[data-testid="stToolbar"] { display: none; }

/* ── Animation ── */
@keyframes fadeUp {
    from { opacity: 0; transform: translateY(12px); }
    to   { opacity: 1; transform: translateY(0); }
}
.main-header  { animation: fadeUp 0.6s ease-out; }
.domain-grid  { animation: fadeUp 0.6s ease-out 0.15s both; }

@media (max-width: 768px) {
    .domain-grid { grid-template-columns: 1fr 1fr; }
    .main-greeting { font-size: 1.6rem; }
}
</style>
""", unsafe_allow_html=True)


# ═══════════════════════════════════════════════════════════════════════
# SIDEBAR
# ═══════════════════════════════════════════════════════════════════════
with st.sidebar:

    # ── Photo ──
```

```
if safe_exists(HEADSHOT_PATH):
    import base64 as _b64
    with open(HEADSHOT_PATH, "rb") as _img_f:
        _b64_str = _b64.b64encode(_img_f.read()).decode()
    _ext = HEADSHOT_PATH.rsplit(".", 1)[-1].lower()
    _mime = {"jpg": "jpeg", "jpeg": "jpeg", "png": "png"}.get(_ext, "jpeg")
    st.markdown(
        f'<img src="{_b64_str}" class="sidebar-phot
        f'alt="Dr. Stephen Dietrich-Kolokouris" />',
        unsafe_allow_html=True,
    )
else:
    st.markdown('<div class="sidebar-photo-placeholder">SDK</div>', unsafe_al

st.markdown('<div class="sidebar-name">Dr. Stephen Dietrich-Kolokouris</div>'
st.markdown('<div class="sidebar-title-line">Cybersecurity · AI/RAG · Intelli

st.markdown("""
<div class="cred-row">
    <span class="cred-tag green">PhD</span>
    <span class="cred-tag green">CCIE</span>
    <span class="cred-tag gold">Ex-CIA Contractor</span>
    <span class="cred-tag blue">Published Author</span>
</div>
""", unsafe_allow_html=True)

st.markdown("---")

# ── Connect ──
st.markdown('<div class="sb-section-title">Connect</div>', unsafe_allow_html=
st.markdown(f"""
<a href="{LINKEDIN_URL}" target="_blank" class="sb-link">↗ LinkedIn</a>
""", unsafe_allow_html=True)

st.markdown("---")

# ── At a Glance ──
st.markdown('<div class="sb-section-title">At a Glance</div>', unsafe_allow_h
st.markdown("""
<div class="sb-item"><strong>Security Architecture</strong> — Pen testing, IR
<div class="sb-item"><strong>AI / RAG Systems</strong> — Production retrieval
<div class="sb-item"><strong>Intelligence</strong> — Former CIA contractor su
<div class="sb-item"><strong>Research & Publishing</strong> — 7 books, PhD (G
""", unsafe_allow_html=True)

st.markdown("---")
```

```
    # ── Conversational toggle ──
    st.session_state.personal_mode = st.toggle(
        "Conversational style",
        value=st.session_state.personal_mode,
        help="Adds career narrative and context to answers.",
    )

    st.markdown("---")

    # ── Publications ──
    with st.expander("Selected Publications"):
        st.markdown("""
        - *The American Paranormal*
        - *Chicago Ripper Crew: Reboot*
        - *Behind the Mask: Hitler the Socialite*
        - *WarSim Algorithm* (DoD submission)
        """)

    # ── About ──
    with st.expander("About This Interface"):
        st.markdown("""
        A retrieval-augmented AI assistant trained on Dr. Dietrich-Kolokouris's
        professional portfolio, publications, and project documentation. Every re
        is grounded in source material.
        """)

    # ── PDF Export ──
    if REPORTLAB_OK and st.session_state.messages:
        st.markdown("---")
        pdf_bytes = build_qa_pdf_bytes(
            st.session_state.messages,
            st.session_state.get("qa_evidence_files", []),
        )
        if pdf_bytes:
            st.download_button(
                label="Export transcript (PDF)",
                data=pdf_bytes,
                file_name="qa_transcript.pdf",
                mime="application/pdf",
                use_container_width=True,
            )


# ═══════════════════════════════════════════════════════════════════════════
# MAIN CONTENT
# ═══════════════════════════════════════════════════════════════════════════
```

```
# ── Header ──
st.markdown("""
<div class="main-header">
    <div class="main-greeting">Ask me anything about Stephen's experience.</div>
    <div class="main-subtitle">
        This assistant draws from a curated portfolio of project documentation, c
        and published research. Describe the role you're filling, and I'll map re
    </div>
</div>
""", unsafe_allow_html=True)

# ── Domain Cards ──
st.markdown("""
<div class="domain-grid">
    <div class="domain-card cyber">
        <span class="domain-icon">🛡</span>
        <div class="domain-label">Security Architecture</div>
        <div class="domain-desc">Pen testing, network audits, CCIE-level infrastr
    </div>
    <div class="domain-card rag">
        <span class="domain-icon">⚙</span>
        <div class="domain-label">AI & RAG Systems</div>
        <div class="domain-desc">Production retrieval pipelines, LangChain, FAISS
    </div>
    <div class="domain-card intel">
        <span class="domain-icon">◈</span>
        <div class="domain-label">Intelligence & Analysis</div>
        <div class="domain-desc">CIA contractor operations, supply chain vulnerab
    </div>
    <div class="domain-card research">
        <span class="domain-icon">✦</span>
        <div class="domain-label">Research & Publishing</div>
        <div class="domain-desc">7 books, PhD in History, WarSim conflict simulat
    </div>
</div>
""", unsafe_allow_html=True)

# ── Chat Section ──
st.markdown('<div class="chat-section-label">Conversation</div>', unsafe_allow_ht

# ── Action Buttons ──
col_a, col_b, col_c = st.columns(3)
with col_a:
    do_verify = st.button("Check sources", use_container_width=True)
with col_b:
    do_fit = st.button("Summarize fit", use_container_width=True)
with col_c:
```

```
    do_outreach = st.button("Draft message", use_container_width=True)

# ── Handle Action Buttons ──
if do_verify:
    last_assistant = None
    for m in reversed(st.session_state.messages):
        if (m.get("role") or "").lower() == "assistant":
            last_assistant = (m.get("content") or "").strip()
            break
    if not last_assistant:
        st.toast("Nothing to check yet.", icon="💬")
    else:
        with st.chat_message("assistant"):
            answer = run_turn("Verify the previous answer:\n\n" + last_assistant,
            st.markdown(answer, unsafe_allow_html=True)
            st.session_state.messages.append({"role": "assistant", "content": ans

if do_fit:
    with st.chat_message("assistant"):
        answer = run_turn(
            "Summarize fit for the role and constraints discussed so far.",
            action_mode="fit",
        )
        st.markdown(answer, unsafe_allow_html=True)
        st.session_state.messages.append({"role": "assistant", "content": answer}

if do_outreach:
    with st.chat_message("assistant"):
        answer = run_turn(
            "Draft an outreach message based on what we've discussed.",
            action_mode="outreach",
        )
        st.markdown(answer, unsafe_allow_html=True)
        st.session_state.messages.append({"role": "assistant", "content": answer}


# ── Pinned Opening ──
if st.session_state.pinned_opening and not st.session_state.messages:
    pinned = (
        "Good to meet you. Tell me about the role you're looking to fill — "
        "I'll walk you through how Stephen's experience lines up, with specifics.
    )
    st.session_state.messages.append({"role": "assistant", "content": pinned})

# ── Render Chat History ──
for m in st.session_state.messages:
    with st.chat_message(m["role"]):
```

```
        st.markdown(m["content"], unsafe_allow_html=True)

# ── Chat Input ──
user_input = st.chat_input("Type a question…")

if user_input:
    st.session_state.messages.append({"role": "user", "content": user_input})
    with st.chat_message("user"):
        st.markdown(user_input)

    with st.chat_message("assistant"):
        answer = run_turn(user_input, action_mode="chat")
        st.markdown(answer, unsafe_allow_html=True)
        st.session_state.messages.append({"role": "assistant", "content": answer}
```