~~~
# QBank Scaffold v7:
# - Integrates full SH calibration into v5 backend
# - Adds RQ job runner + progress tracking
# - Admin UI progress screen
# - Unit tests for SH estimator (seeded synthetic pool)
# - Per-topic caps & blueprint-weighted τ
#
# Output: /mnt/data/qbank_scaffold_v7.zip

import os, zipfile, pathlib, textwrap, json

ROOT = "/mnt/data/qbank_scaffold_v7"
os.makedirs(ROOT, exist_ok=True)

def write(path, content):
    os.makedirs(os.path.dirname(path), exist_ok=True)
    with open(path, "w", encoding="utf-8") as f:
        f.write(textwrap.dedent(content))

# ---------- README ----------
write(f"{ROOT}/README.md", """
# QBank Scaffold (v7) — Integrated SH, Jobs, Progress UI, Topic Caps

What’s new on top of v5b/v6:
- **Full Sympson–Hetter** iterative calibration integrated into the backend.
- **RQ job runner** (Redis) to run calibrations async, with progress updates.
- **Admin UI** page to launch calibrations and watch progress live.
- **Unit tests** for the SH estimator with a seeded synthetic pool.
- **Per-topic caps or blueprint-weighted τ**: calibrate using a scalar τ, a per-topic
τ map, or topic weights.

## Quick start
1) Services (Postgres + Redis) up; DB schema from previous versions +
`item_exposure_control` present.
2) Backend API:
```bash
cd qbank-backend
pip install -r requirements.txt
uvicorn app.main:app --reload
Worker:

bash
~~~

Always show details

```
Copy
cd qbank-backend
python -m app.jobs.worker
Admin UI:

bash
Always show details

Copy
cd admin-ui && npm install && npm run dev   # http://localhost:4000
Launch a calibration from the UI (Admin → Calibration) or via API:

bash
Always show details

Copy
POST /v1/admin/exposure/calibrate_sh/start
{
  "exam_code":"DEMO-EXAM", "tau":0.2, "n":400, "test_len":25, "iters":5, "alpha":0.6,
  "topic_tau": {"123":0.18, "124":0.22},                    // optional
  "topic_weights": {"123":2.0, "124":1.0},                  // optional
  "dry_run": false
}
Cron / scheduled runs
Kubernetes: see scripts/cron_k8s/calibration-cronjob.yaml.

GitHub Actions (example only): .github/workflows/nightly_calibration.yml.

Notes
Progress is stored in RQ job meta: {current_iter, total_iters, avg_exp, max_over}
and exposed via /status endpoint.

If both topic_tau and topic_weights are passed, topic_tau takes precedence;
otherwise weights are normalized to a τ per topic.
""")

---------- Requirements ----------
write(f"{ROOT}/qbank-backend/requirements.txt", """
fastapi==0.115.0
uvicorn==0.30.6
pydantic==2.8.2
python-dotenv==1.0.1
```

```
redis==5.0.8
rq==1.16.2
kafka-python==2.0.2
psycopg2-binary==2.9.9
SQLAlchemy==2.0.32
PyJWT==2.9.0
black==24.8.0
flake8==7.1.1
pytest==8.3.2
httpx==0.27.2
""")

---------- Core config (add RQ settings) ----------
write(f"{ROOT}/qbank-backend/app/core/config.py", """
import os
from dotenv import load_dotenv
load_dotenv()

DATABASE_URL = os.getenv("DATABASE_URL", "postgresql+psycopg2://
qbank:qbank@localhost:5432/qbank")
REDIS_URL = os.getenv("REDIS_URL", "redis://localhost:6379/0")

RQ
RQ_QUEUE = os×getenv("RQ_QUEUE", "calibration")
RQ_WORKER_CONCURRENCY = int(os.getenv("RQ_WORKER_CONCURRENCY",
"1"))

KAFKA_BOOTSTRAP = os.getenv("KAFKA_BOOTSTRAP", "localhost:9092")
KAFKA_TOPIC_EVENTS = os.getenv("KAFKA_TOPIC_EVENTS", "events.qbank")
TENANT_ID = os.getenv("APP_TENANT_ID",
"00000000-0000-0000-0000-000000000001")
APP_SECRET = os.getenv("APP_SECRET", "dev-secret-change-me")
MAX_DAILY_EXPOSURES = int(os.getenv("MAX_DAILY_EXPOSURES", "500"))
""")

write(f"{ROOT}/qbank-backend/app/core/cache.py", """
import redis
from datetime import datetime
from app.core.config import REDIS_URL, MAX_DAILY_EXPOSURES

redis_client = redis.Redis.from_url(REDIS_URL, decode_responses=True)

def exposure_key(question_id: int, version: int) -> str:
day = datetime×utcnow()×strftime("%Y%m%d")
```

```
return f"exp:{day}:{question_id}:{version}"

def can_serve(question_id: int, version: int) -> bool:
key = exposure_key(question_id, version)
count = int(redis_client.get(key) or 0)
return count < MAX_DAILY_EXPOSURES

def bump_exposure(question_id: int, version: int) -> None:
key = exposure_key(question_id, version)
pipe = redis_client.pipeline()
pipe.incr(key, 1)
pipe.expire(key, 86400)
pipe.execute()
""")

---------- SH core (library) ----------
write(f"{ROOT}/analytics/calibration/sh_core.py", """
import math, random, statistics
import psycopg2, psycopg2.extras

D = 1.7
def logistic(x: float) -> float: return 1.0 / (1.0 + math.exp(-x))
def prob_3pl(theta: float, a: float, b: float, c: float) -> float:
return c + (1.0 - c) * logistic(D * a * (theta - b))
def fisher_info_3pl(theta: float, a: float, b: float, c: float) -> float:
P = prob_3pl(theta, a, b, c); Q = 1.0 - P
if P<=0 or Q<=0 or (1.0-c)<=0: return 0.0
return (D2)*(a2)(Q/P)((P-c)/(1.0-c))**2

def sample_theta(dist: str = "normal0,1"):
if dist.startswith("normal"): return random.gauss(0.0, 1.0)
if dist.startswith("uniform"): return random.uniform(-1.0, 1.0)
return 0.0

def load_pool(conn, exam_code: str):
sql = '''
SELECT qv.question_id, qv.version, qv.topic_id,
COALESCE(ic.a, 1.0) AS a, COALESCE(ic.b, 0.0) AS b, COALESCE(ic.c, 0.2) AS c,
COALESCE(iec.sh_p, 1.0) AS sh_p
FROM question_publications qp
JOIN question_versions qv ON qv.question_id=qp.question_id AND
qv.version=qp.live_version
LEFT JOIN item_calibration ic ON ic.question_id=qv.question_id AND
ic.version=qv.version AND ic.model='3pl'
```

```
LEFT JOIN item_exposure_control iec ON iec.question_id=qv.question_id AND
iec.version=qv.version
WHERE qp.exam_code=%s AND qv.state='published'
'''
with conn.cursor(cursor_factory=psycopg2.extras.DictCursor) as cur:
cur.execute(sql, (exam_code,))
rows = cur.fetchall()
items = []
for r in rows:
items.append({
"qid": int(r["question_id"]), "ver": int(r["version"]),
"topic": r["topic_id"], "a": float(r["a"]), "b": float(r["b"]), "c": float(r["c"]),
"k": float(r["sh_p"]) if r["sh_p"] is not None else 1.0
})
return items

def simulate_once(pool, n: int, test_len: int, theta_dist: str, use_k: bool = True,
seed: int | None = None):
if seed is not None: random.seed(seed)
seen = { (it["qid"], it["ver"]): 0 for it in pool }
proposed = { (it["qid"], it["ver"]): 0 for it in pool }
```

python

Always show details

Copy

```
for _ in range(n):
    theta = sample_theta(theta_dist)
    administered = set()
    for _pos in range(test_len):
        candidates = [it for it in pool if (it["qid"], it["ver"]) not in administered]
        if not candidates: break
        scored = [(fisher_info_3pl(theta, it["a"], it["b"], it["c"]), it) for it in
candidates]
        scored×sort(key=lambda x: x[0], reverse=True)
        chosen = None
        for _, it in scored:
            key = (it["qid"], it["ver"])
            proposed[key] += 1
            if use_k:
                if random.random() <= max(0.0, min(1.0, it["k"])):
                    chosen = it; break
                else:
                    continue
            else:
                chosen = it; break
        if chosen is None: chosen = scored[0][1]
        key = (chosen["qid"], chosen["ver"])
        administered.add(key)
    for key in administered: seen[key] += 1
return seen, proposed
def _compute_topic_tau(tau: float, topic_tau: dict | None, topic_weights: dict |
None) -> dict | None:
if topic_tau: return {str(k): float(v) for k,v in topic_tau.items()}
if topic_weights:
# Normalize weights to sum to 1 and scale by tau
s = sum(float(v) for v in topic_weights.values()) or 1.0
return {str(k): tau * (float(v)/s) for k,v in topic_weights.items()}
return None

def iterative_sh(pool, tau: float, n: int, test_len: int, iters: int, alpha: float,
theta_dist: str, floor: float, ceil: float, seed: int | None = None,
topic_tau: dict | None = None, topic_weights: dict | None = None):
topic_tau_map = _compute_topic_tau(tau, topic_tau, topic_weights)
k = { (it["qid"], it["ver"]): float(it["k"]) for it in pool }
history = []
for t in range(iters):
for it in pool: it["k"] = k[(it["qid"], it["ver"])]
seen, _ = simulate_once(pool, n=n, test_len=test_len, theta_dist=theta_dist,
use_k=True, seed=(None if seed is None else seed + t))
r = { key: seen[key] / max(1, n) for key in seen }
newk, diffs = {}, []
for it in pool:
key = (it["qid"], it["ver"])
ri = r×get(key, 0.0); ki = k[key]
# choose cap
cap = tau
if topic_tau_map is not None:
cap = float(topic_tau_map.get(str(it["topic"]), tau))
if ri <= 0.0:
newk[key] = min(1.0, max(floor, ki * 1.1))
else:
ratio = cap / ri
adj = ratio ** alpha
cand = ki × adj
newk[key] = min(1.0, max(floor, min(ceil, cand)))
diffs.append(abs(ri - min(cap, ri)))
k = newk
```

```
avg_exp = (sum(r×values())/len(r)) if r else 0.0
max_over = 0.0
if topic_tau_map is None:
max_over = max((ri - tau for ri in r.values()), default=0.0)
else:
# compute per-topic overage
overages = []
for it in pool:
key = (it["qid"], it["ver"]); cap = float(topic_tau_map.get(str(it["topic"]), tau))
overages.append(r.get(key,0.0) - cap)
max_over = max(overages) if overages else 0.0
history.append({"iter": t+1, "avg_exp": avg_exp, "max_over": max_over})
return k, seen, history

def upsert_k(conn, kmap):
with conn.cursor() as cur:
cur.execute("SET search_path TO public")
for (qid, ver), kval in kmap.items():
cur.execute(
'''INSERT INTO item_exposure_control(question_id, version, sh_p)
VALUES (%s,%s,%s)
ON CONFLICT (question_id,version) DO UPDATE SET sh_p=EXCLUDED.sh_p,
updated_at=now()''',
(qid, ver, float(kval))
)
conn.commit()
""")

---------- CLI (wraps core) ----------
write(f"{ROOT}/analytics/calibration/sh_iterative.py", """
import argparse, json, psycopg2
from .sh_core import load_pool, iterative_sh, upsert_k

def main():
ap = argparse×ArgumentParser()
ap.add_argument("--dsn", required=True)
ap.add_argument("--exam", "--exam_code", dest="exam_code", required=True)
ap.add_argument("--tau", type=float, default=0.2)
ap.add_argument("--n", type=int, default=500)
ap.add_argument("--len", "--test_len", dest="test_len", type=int, default=30)
ap.add_argument("--iters", type=int, default=6)
ap.add_argument("--alpha", type=float, default=0.6)
ap.add_argument("--theta", "--theta_dist", dest="theta_dist",
default="normal0,1")
```

```
ap.add_argument("--floor", type=float, default=0.02)
ap.add_argument("--ceil", type=float, default=1.0)
ap.add_argument("--seed", type=int, default=None)
ap.add_argument("--topic_tau", type=str, help="JSON {topic_id: tau_cap}",
default=None)
ap.add_argument("--topic_weights", type=str, help="JSON {topic_id: weight}",
default=None)
ap.add_argument("--dry_run", action="store_true")
args = ap×parse_args()
```

lua
Always show details

Copy

```
topic_tau = json×loads(args×topic_tau) if args.topic_tau else None
topic_weights = json.loads(args.topic_weights) if args.topic_weights else None

conn = psycopg2.connect(args×dsn)
pool = load_pool(conn, args.exam_code)
if not pool:
    print("No items found for", args.exam_code); return

kmap, seen, hist = iterative_sh(
    pool, tau=args×tau, n=args×n, test_len=args.test_len, iters=args.iters,
    alpha=args×alpha, theta_dist=args.theta_dist, floor=args×floor, ceil=args×ceil,
    seed=args×seed, topic_tau=topic_tau, topic_weights=topic_weights
)
print("history=", hist)
if not args.dry_run:
    upsert_k(conn, kmap)
    print("upserted=", len(kmap))
conn.close()
if name == "main":
main()
""")

---------- Jobs: RQ queue + worker + job function ----------
write(f"{ROOT}/qbank-backend/app/jobs/queue.py", """
from rq import Queue
from redis import Redis
from app.core.config import REDIS_URL, RQ_QUEUE

redis = Redis.from_url(REDIS_URL)
queue = Queue(RQ_QUEUE, connection=redis)
```

```
""")

write(f"{ROOT}/qbank-backend/app/jobs/calibration_job.py", """
import os, json, psycopg2
from rq import get_current_job
from analytics.calibration.sh_core import load_pool, iterative_sh, upsert_k

def calibrate_job(exam_code: str, dsn: str, tau: float, n: int, test_len: int, iters: int,
alpha: float, theta_dist: str, floor: float, ceil: float,
topic_tau: dict | None, topic_weights: dict | None, dry_run: bool):
job = get_current_job()
job.meta.update({"state": "running", "current_iter": 0, "total_iters": iters});
job.save_meta()
```

python
Always show details

Copy

```
conn = psycopg2.connect(dsn)
pool = load_pool(conn, exam_code)
if not pool:
    job.meta.update({"state":"empty"}); job.save_meta(); conn.close(); return
{"updated":0,"history":[]}

# iterative with callback-like progress (loop inside iterative_sh can't call back
easily, so call per-iter here)
kmap = { (it["qid"], it["ver"]): float(it["k"]) for it in pool }
history = []
for t in range(iters):
    # run one-iter by setting iters=1 and seeding changes each loop
    km, seen, hist = iterative_sh(pool, tau=tau, n=n, test_len=test_len, iters=1,
alpha=alpha,
                                 theta_dist=theta_dist, floor=floor, ceil=ceil, seed=None,
                                 topic_tau=topic_tau, topic_weights=topic_weights)
    # update pool ks
    for it in pool: it["k"] = km[(it["qid"], it["ver"])]
    kmap = km
    history.extend(hist)
    job.meta.update({"current_iter": t+1, "avg_exp": hist[-1]["avg_exp"],
"max_over": hist[-1]["max_over"]}); job.save_meta()

if not dry_run:
    upsert_k(conn, kmap)
conn.close()
```

```
job.meta.update({"state":"done"}); job.save_meta()
return {"updated": len(kmap), "history": history}
""")

write(f"{ROOT}/qbank-backend/app/jobs/worker.py", """
from rq import Worker
from app.jobs.queue import queue, redis
from app.core.config import RQ_QUEUE, RQ_WORKER_CONCURRENCY

if name == "main":
# Note: RQ doesn't handle concurrency within a single worker process; scale by processes
w = Worker([RQ_QUEUE], connection=redis)
w.work(with_scheduler=True)
""")

---------- Admin API additions: start/status ----------
write(f"{ROOT}/qbank-backend/app/api/admin.py", """
from fastapi import APIRouter, Depends, HTTPException
from pydantic import BaseModel, Field
from typing import List, Optional
from sqlalchemy.orm import Session
from sqlalchemy import text
import os, json
from rq.job import Job
from app.core.database import get_db
from app.core.auth import require_roles
from app.core.config import REDIS_URL
from app.jobs.queue import queue, redis
from app.jobs.calibration_job import calibrate_job
from app.models.orm import ItemExposureControl

router = APIRouter()

class ItemRow(BaseModel):
question_id: int
version: int
topic_id: Optional[int] = None
sh_p: float
recent_attempts: int | None = 0

@router.get("/exposure/items", response_model=List[ItemRow], dependencies=[Depends(require_roles("admin"))])
def list_items(limit: int = 100, db: Session = Depends(get_db)):
```

```
rows = db.execute(text("""
SELECT qv.question_id, qv.version, qv.topic_id,
COALESCE(iec.sh_p, 1.0) as sh_p,
(SELECT count(*) FROM user_responses ur WHERE ur.question_id=qv.question_id AND ur.version=qv.version AND ur.created_at>now()-interval '7 days') as recent_attempts
FROM question_versions qv
LEFT JOIN item_exposure_control iec ON iec.question_id=qv.question_id AND iec.version=qv.version
WHERE qv.state='published'
ORDER BY recent_attempts DESC NULLS LAST
LIMIT :lim
"""), {"lim": limit}).all()
return [ItemRow(question_id=r[0], version=r[1], topic_id=r[2], sh_p=float(r[3]), recent_attempts=r[4] or 0) for r in rows]

class SetSh(BaseModel):
question_id: int
version: int
sh_p: float

@router.post("/exposure/set", dependencies=[Depends(require_roles("admin"))])
def set_sh(payload: SetSh, db: Session = Depends(get_db)):
if payload.sh_p < 0 or payload.sh_p > 1: raise HTTPException(400, "sh_p must be in [0,1]")
row = db.get(ItemExposureControl, {"question_id": payload.question_id, "version": payload.version})
if row: row.sh_p = payload.sh_p
else: db.add(ItemExposureControl(question_id=payload.question_id, version=payload×version, sh_p=payload.sh_p))
db.commit()
return {"ok": True}

class StartCalib(BaseModel):
exam_code: str
tau: float = 0.2
n: int = 400
test_len: int = 25
iters: int = 5
alpha: float = 0.6
theta_dist: str = "normal0,1"
floor: float = 0.02
ceil: float = 1.0
topic_tau: Optional[dict] = None
```

```
topic_weights: Optional[dict] = None
dry_run: bool = False

@router.post("/exposure/calibrate_sh/start",
dependencies=[Depends(require_roles("admin"))])
def start_calibration(payload: StartCalib):
dsn = os.getenv("DATABASE_URL", "postgresql+psycopg2://
qbank:qbank@localhost:5432/qbank")
job = queue.enqueue(
calibrate_job,
payload.exam_code, dsn, payload.tau, payload.n, payload.test_len, payload.iters,
payload.alpha, payload.theta_dist, payload.floor, payload.ceil,
payload.topic_tau, payload.topic_weights, payload.dry_run,
job_timeout=3600
)
return {"job_id": job.get_id()}

class CalibStatus(BaseModel):
state: str
current_iter: int
total_iters: int
avg_exp: float | None = None
max_over: float | None = None
result: Optional[dict] = None

@router.get("/exposure/calibrate_sh/status", response_model=CalibStatus,
dependencies=[Depends(require_roles("admin"))])
def calib_status(job_id: str):
job = Job×fetch(job_id, connection=redis)
meta = job×meta or {}
state = meta×get("state") or job.get_status()
return CalibStatus(
state=state,
current_iter=int(meta×get("current_iter") or 0),
total_iters=int(meta×get("total_iters") or 0),
avg_exp=float(meta×get("avg_exp") or 0.0) if meta.get("avg_exp") is not None
else None,
max_over=float(meta.get("max_over") or 0.0) if meta.get("max_over") is not None
else None,
result=job.result if state == "done" else None
)
""")

---------- Main app (integrated) ----------
```

```
write(f"{ROOT}/qbank-backend/app/main.py", """
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.api.quizzes import router as quizzes_router
from app.api.author import router as author_router
from app.api.auth import router as auth_router
from app.api.admin import router as admin_router

app = FastAPI(title="QBank API v7", version="7.0.0")
app.add_middleware(CORSMiddleware, allow_origins=[""],
allow_credentials=True, allow_methods=[""], allow_headers=["*"])

app.include_router(auth_router, prefix="/v1/auth", tags=["auth"])
app.include_router(quizzes_router, prefix="/v1/quizzes", tags=["quizzes"])
app.include_router(author_router, prefix="/v1/author", tags=["authoring"])
app.include_router(admin_router, prefix="/v1/admin", tags=["admin"])

@app.get("/health")
def health(): return {"status": "ok"}
""")

---------- Admin UI: calibration progress page ----------
write(f"{ROOT}/admin-ui/pages/calibration.tsx", """
import { useEffect, useState } from 'react';
const API = process.env.NEXT_PUBLIC_API || 'http://localhost:8000';

type Status = { state:string; current_iter:number; total_iters:number;
avg_exp?:number; max_over?:number; result?:any };

export default function Calibration() {
const [token, setToken] = useState('');
const [jobId, setJobId] = useState<string>('');
const [status, setStatus] = useState<Status | null>(null);
const [form, setForm] = useState({ exam_code:'DEMO-EXAM', tau:0.2, n:400,
test_len:25, iters:5, alpha:0.6 });

const headers = { 'Content-Type':'application/json', 'Authorization': Bearer $
{token} };

const start = async () => {
const r = await fetch(${API}/v1/admin/exposure/calibrate_sh/start,
{ method:'POST', headers, body: JSON.stringify(form) });
const data = await r.json(); setJobId(data.job_id);
};
```

```
useEffect(()=>{
const t = setInterval(async () => {
if (!jobId) return;
const r = await fetch(${API}/v1/admin/exposure/calibrate_sh/status?job_id=$
{jobId}, { headers });
if (r.ok) setStatus(await r.json());
}, 1500);
return () => clearInterval(t);
}, [jobId, token]);

const pct = status?.total_iters ? Math.round(100 * (status!.current_iter /
status!.total_iters)) : 0;

return (
<main style={{padding:24, maxWidth:800}}>
<h1>Calibration</h1>
<p>Paste an <b>admin</b> JWT</p>
<textarea value={token} onChange={(e)=>setToken(e.target.value)} rows={3}
style={{width:'100%'}} />
<section style={{marginTop:16}}>
<h3>Parameters</h3>
<div style={{display:'grid', gridTemplateColumns:'repeat(3, 1fr)', gap:12}}>
<label>Exam <input value={form.exam_code} onChange={(e)=>setForm({...form,
exam_code:e.target.value})} /></label>
<label>τ <input type="number" step="0.01" value={form.tau}
onChange={(e)=>setForm({...form, tau:parseFloat(e.target.value)})} /></label>
<label>n <input type="number" value={form.n} onChange={(e)=>setForm({...form,
n:parseInt(e.target.value)})} /></label>
<label>length <input type="number" value={form.test_len}
onChange={(e)=>setForm({...form, test_len:parseInt(e.target.value)})} /></label>
<label>iters <input type="number" value={form.iters}
onChange={(e)=>setForm({...form, iters:parseInt(e.target.value)})} /></label>
<label>α <input type="number" step="0.1" value={form.alpha}
onChange={(e)=>setForm({...form, alpha:parseFloat(e.target.value)})} /></label>
</div>
<button style={{marginTop:12}} onClick={start}>Start</button>
</section>
{status && (
<section style={{marginTop:24}}>
<h3>Status: {status.state}</h3>
<div style={{height:16, background:'#eee', borderRadius:8, overflow:'hidden'}}>
<div style={{width:${pct}%, height:'100%', background:'#4a90e2'}} />
</div>
<p style={{marginTop:8}}>iter {status.current_iter} / {status.total_iters} · avg_exp
{status.avg_exp?.toFixed(3)} · max_over {status.max_over?.toFixed(3)}</p>
{status.result && <pre style={{maxHeight:200, overflow:'auto',
background:'#fafafa', padding:12}}>{JSON.stringify(status.result, null, 2)}</pre>}
</section>
)}
</main>
);
}
""")

---------- Unit tests for SH core ----------
write(f"{ROOT}/analytics/tests/test_sh_core.py", """
from analytics.calibration.sh_core import iterative_sh

def test_iterative_sh_converges_under_tau():
# Build a tiny synthetic pool with varied difficulty/discrimination
pool = [
{"qid":1,"ver":1,"topic":101,"a":1.2,"b":0.0,"c":0.2,"k":1.0},
{"qid":2,"ver":1,"topic":101,"a":0.8,"b":-0.5,"c":0.2,"k":1.0},
{"qid":3,"ver":1,"topic":102,"a":1.0,"b":0.5,"c":0.2,"k":1.0},
{"qid":4,"ver":1,"topic":102,"a":1.6,"b":0.0,"c":0.2,"k":1.0},
]
tau = 0.25
kmap, seen, hist = iterative_sh(pool, tau=tau, n=200, test_len=8, iters=4,
alpha=0.7, theta_dist="normal0,1", floor=0.02, ceil=1.0, seed=42)
# final iteration summary must exist
assert hist[-1]["iter"] == 4
# all k are within bounds
assert all(0.02 <= v <= 1.0 for v in kmap.values())
""")

---------- Cron: K8s example & GitHub Actions example ----------
write(f"{ROOT}/scripts/cron_k8s/calibration-cronjob.yaml", """
apiVersion: batch/v1
kind: CronJob
metadata:
name: qbank-calibration
spec:
schedule: "0 3 * * *"
jobTemplate:
spec:
template:
spec:
```

```
restartPolicy: Never
containers:
- name: calibrate
image: yourrepo/qbank-backend:latest
env:
- name: DATABASE_URL
valueFrom: { secretKeyRef: { name: qbank-secrets, key: DATABASE_URL } }
command: ["python","analytics/calibration/sh_iterative.py"]
args: ["--dsn","$(DATABASE_URL)","--exam","DEMO-EXAM","--tau","0.2","--n","1000","--len","30","--iters","6","--alpha","0.6"]
""")

write(f"{ROOT}/.github/workflows/nightly_calibration.yml", """
name: nightly-calibration
on:
schedule:
- cron: "0 4 * * *"
jobs:
run:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v4
- uses: actions/setup-python@v5
with: { python-version: "3.11" }
- name: Install deps
run: pip install -r qbank-backend/requirements.txt
- name: Run calibration (requires network access to DB)
env:
DATABASE_URL: ${{ secrets.QBANK_DATABASE_URL }}
run: |
python analytics/calibration/sh_iterative.py --dsn "$DATABASE_URL" --exam DEMO-EXAM --tau 0.2 --n 1000 --len 30 --iters 6 --alpha 0.6 --dry_run
""")

---------- Zip ----------
zip_path = "/mnt/data/qbank_scaffold_v7.zip"
with zipfile.ZipFile(zip_path, "w", zipfile.ZIP_DEFLATED) as z:
for p in pathlib.Path(ROOT).rglob("*"):
z.write(p, p.relative_to(ROOT))

zip_path
```

Always show details

Copy
Result
'/mnt/data/qbank_scaffold_v7.zip'
Integrated and upgraded. Here's your bundle:

Download v7 (integrated SH + jobs + UI + tests)

What's in this cut:

Full SH inside your v5 backend

Refactor into a library: analytics/calibration/sh_core.py (simulation + iterative updater + upsert).

CLI wrapper: analytics/calibration/sh_iterative.py still available for ad-hoc/cron.

Async calibration via RQ

Queue/worker: qbank-backend/app/jobs/{queue.py, worker.py}.

Job fn updates progress in job.meta each iteration.

Admin API:

POST /v1/admin/exposure/calibrate_sh/start → enqueues with params (τ, n, len, iters, α… plus per-topic settings).

GET /v1/admin/exposure/calibrate_sh/status?job_id= → live progress (iter, avg exposure, max over-cap) + result on completion.

Progress UI for admins

admin-ui/pages/calibration.tsx: form to launch runs, progress bar with polling, result preview.

Per-topic caps & blueprint-weighted τ

Pass either topic_tau (dict of topic_id → τ) or topic_weights (dict of weights). If both given, explicit caps win; otherwise weights are normalized to produce τ per topic.

Unit tests

analytics/tests/test_sh_core.py: synthetic seed; verifies bounds & iteration flow. Extend easily with assertions on convergence if desired.

Scheduling examples

Kubernetes CronJob: scripts/cron_k8s/calibration-cronjob.yaml.

GitHub Actions nightly (example/dry-run): .github/workflows/nightly_calibration.yml.

Spin-up quick:

API + worker

bash
Copy
Edit

```
cd qbank-backend
pip install -r requirements.txt
uvicorn app.main:app --reload   # terminal A
python -m app.jobs.worker       # terminal B
```

Admin UI

arduino
Copy
Edit

```
cd admin-ui
npm install
npm run dev   # http://localhost:4000/calibration
```

Launch a job (UI or cURL)

bash
Copy
Edit

```
POST /v1/admin/exposure/calibrate_sh/start
{
  "exam_code":"DEMO-EXAM","tau":0.2,"n":400,"test_len":25,"iters":5,"alpha":0.6,
  "topic_weights":{"101":2,"102":1}, "dry_run": false
}
```