Firebase AI Blog Pipeline — Modular Design & Scaffolding

This document translates your Colab workflow into a modular, scalable set of Firebase Cloud Functions / Genkit flows. It includes module interfaces, prompt templates, Firestore schemas, and security/IaC notes. When you paste your Colab code, we'll map each block line-by-line into these modules and remove any redundant logic now handled by infrastructure.

---

0) High-Level Architecture

Triggering options - Manual run via HTTPS callable (`/orchestrate:runOnce`). - Scheduled run (Cloud Scheduler → Pub/Sub → Orchestrator). - Firestore event-driven (create a `pipeline_runs/{runId}` doc with fields: `topicSeed`, `status`, etc.).

Execution pattern - **Orchestrator** writes a `runs/{runId}` doc and enqueues sequential rounds as discrete functions. Each round updates `status`, `artifacts`, and `diagnostics` on the run doc. - **Idempotency**: every round checks for an existing output artifact on the run doc before re-computation. - **Caching**: SerpApi results cached under `cache/serpapi/{hash}` and reused for 24–48h. - **Observability**: structured logs per round with `runId`, `round`, `durationMs`, `tokenUsage`.

Core collections - `runs/{runId}` — state machine for a pipeline execution. - `topics/{topicId}` — canonical topics (deduplicated via embeddings). - `drafts/{draftId}` — assembled post drafts ready to publish. - `cache/serpapi/{queryHash}` — cached SERP/autocomplete/trends payloads. - `vectors/{embeddingId}` — optional lightweight store for embeddings & metadata.

---

1) Round Mapping (Colab → Modules)

> We will plug your pasted Colab blocks here. Pre-mapping guidance below:

Colab Capability	Keep	Simplify	Remove/Replace
Feed fetching / trends via custom requests	Keep behind `serpApiClient` with caching	Yes: normalize payload into `TrendItem[]`	Replace ad-hoc parsing with SerpApi client responses
Clustering (topic grouping)	Keep if lightweight	Use embeddings + simple agglomerative clustering	Replace heavy libs with sentence-transformers and small thresholds
Duplicate detection	Keep core logic	Replace heuristics with embedding cosine similarity	Remove exact string dedup if embeddings used
WordPress posting	Keep core	Wrap in `wpClient` with retries & draft status	Remove inline auth logic; use Secret Manager

Colab Capability	Keep	Simplify	Remove/Replace
Local CSV/JSON writes	Optional	Mirror into Firestore `artifacts` field	Remove filesystem assumptions
Notebook-only progress prints	Remove	Use structured logging (`functions.logger`)	Yes

Once you paste the script, we'll annotate each cell and map it to the appropriate round function.

---

2) Module Interfaces (Inputs/Outputs)

Shared types (TypeScript):

```
export type RunId = string;
export interface TrendItem { query: string; type:
"autocomplete"|"trending"|"related"; score?: number; region?: string;
source: string; ts: string; }
export interface TopicIdea { title: string; rationale: string; seed:
string; }
export interface OutlineSection { id: string; heading: string; bullets:
string[]; estWords?: number; }
export interface Outline { title: string; sections: OutlineSection[]; }
export interface SectionDraft { sectionId: string; content: string;
citations?: string[]; }
export interface PolishedSection { sectionId: string; content: string;
readability: { fkGrade?: number }; }
export interface Metadata { seoTitle: string; metaDescription: string; tags:
string[]; slug: string; }
export interface ImagePrompt { prompt: string; negative?: string[]; altText:
string; }
export interface CoherenceReport { overall: number; duplicates:
Array<{againstId:string; score:number}>; notes: string[]; }
export interface WPDraft { postId?: number; status: "draft"|"publish";
editUrl?: string; }
```

Round 0 — Trend & Topic Input

- **fn**: `round0FetchTrends(runId: RunId, seeds: string[], region?: string)`
- **in**: `seeds` (free text), `region` (e.g., `in`, `us`)
- **out**: `TrendItem[]`
- **side effects**: caches `TrendItem[]` in `cache/serpapi/{hash}` and writes to `runs/{runId}.artifacts.round0`

Round 1 — Topic Ideation (TinyLlama)

- **fn**: `round1IdeateTopics(runId, trends: TrendItem[]): TopicIdea[]`
- **out**: 3–5 `TopicIdea` with rationales

Round 2 — Outline Generation (Phi-3 Mini)

- **fn**: `round2Outline(runId, idea: TopicIdea): Outline`

Round 3 — Section Drafting (Mistral Small 3.1 or similar)

- **fn**: `round3DraftSections(runId, outline: Outline): SectionDraft[]`

Round 4 — Tone Polishing (LLaMA 3.1 8B)

- **fn**: `round4Polish(runId, drafts: SectionDraft[], brandVoice: string): PolishedSection[]`

Round 5 — Metadata & Image Prompts (Gemma 2–7B or TinyLlama)

- **fn**: `round5Metadata(runId, outline: Outline, polished: PolishedSection[]): { meta: Metadata; images: ImagePrompt[] }`

Round 6 — Coherence & Duplication Checks (Embeddings)

- **fn**: `round6Coherence(runId, texts: string[]): CoherenceReport`
- **side effects**: upsert `vectors/*` and de-duplicate against `topics/*` and recent `drafts/*`

Round 7 — Publish to WordPress

- **fn**: `round7Publish(runId, polished: PolishedSection[], meta: Metadata): WPDraft`

---

3) Prompt Templates (per Round)

> Design notes: short, explicit, role-based. Use fenced sections, require placeholders for citations (e.g., `[source?]`). Add examples where helpful.

Round 1 — TinyLlama (Topic Ideation)

```
SYSTEM: You are a concise content strategist. You propose practical blog post
titles based on trend signals. Avoid speculation; request citations as
placeholders.
TASK: Given TREND_SIGNALS, produce 3–5 titles with one-sentence rationales.
Titles must be specific, useful, non-clickbait.
CONSTRAINTS:
- Include a primary keyword from input.
- Business-neutral tone.
- Each rationale ends with `[source?]`.
INPUT/TREND_SIGNALS:
{{json(trendItems)}}
OUTPUT JSON SCHEMA:
{
  "ideas": [
    { "title": "string", "rationale": "string", "seed": "string" }
```

```
  ]
}
```

Round 2 — Phi-3 Mini (Outline)

```
SYSTEM: You are a technical editor. Build a structured outline for a blog
post.
TASK: Using TOPIC_IDEA, create 5–8 sections with headings, 3–5 bullets each,
and estimated word counts.
STYLE: Clear, skimmable, includes a short intro and conclusion.
INPUT/TOPIC_IDEA:
{{json(topicIdea)}}
OUTPUT JSON SCHEMA:
{ "title": "string", "sections": [
  { "id": "s1", "heading": "string", "bullets": ["string"], "estWords": 120 }
]}
```

Round 3 — Mistral Small 3.1 (Draft Sections)

```
SYSTEM: You are a subject-matter writer. Expand each outline section into a
coherent draft. Use neutral, accurate language.
TASK: For each section, write 120–220 words. Add inline citation placeholders
like [1], [2] where claims need support.
STYLE: Short paragraphs, active voice, concrete examples.
INPUT/OUTLINE:
{{json(outline)}}
OUTPUT JSON SCHEMA:
[ { "sectionId": "s1", "content": "... [1] ..." } ]
```

Round 4 — LLaMA 3.1 8B (Tone Polishing)

```
SYSTEM: You are a copy editor enforcing brand voice.
BRAND_VOICE: {{brandVoice}}
TASK: Improve readability (target grade 8–10), maintain facts, keep citation
placeholders intact.
CONSTRAINTS: Do not add new claims. Keep section boundaries.
INPUT/DRAFTS:
{{json(sectionDrafts)}}
OUTPUT JSON SCHEMA:
[ { "sectionId": "s1", "content": "string", "readability": { "fkGrade":
9.1 } } ]
```

Round 5 — Gemma 2–7B (Metadata & Images)

```
SYSTEM: You are an SEO and art-direction assistant.
TASKS:
```

```
1) Create SEO title (<= 60 chars), meta description (140–160), slug, and 6–10
tags.
2) Propose 2–3 image generation prompts (photorealistic or illustration).
Include alt text.
INPUT: {{json({title:outline.title, sections:outline.sections})}}
OUTPUT JSON SCHEMA:
{
  "meta": { "seoTitle": "", "metaDescription": "", "slug": "", "tags": [] },
  "images": [ { "prompt": "", "negative": [""], "altText": "" } ]
}
```

Round 6 — Embedding Coherence

```
ALGORITHM NOTE (not a prompt):
- Model: sentence-transformers/all-MiniLM-L6-v2 (or e5-small).
- Compute embeddings for: title, each section, and concatenated post.
- Coherence = average cosine(section, post) and adjacency similarity.
- Duplicate check: compare title/post vector to last N=500 posts in Firestore
vectors; flag if cosine > 0.92.
```

Round 7 — WordPress Publish

```
No LLM. Assemble: H1 = title; H2 per section; paragraphs under each. Append
references section reserved for citations. Post as `status=draft`.
```

---

4) Firestore Schema (documents)

`runs/{runId}`

```
{
  "createdAt": "serverTimestamp",
  "status": "running|error|done",
  "round": 0,
  "params": { "seeds": ["ai tools"], "region": "in", "brandVoice":
"practical, friendly" },
  "artifacts": {
    "round0": { "trendItems": [/* TrendItem */] },
    "round1": { "ideas": [/* TopicIdea */] },
    "round2": { "outline": {/* Outline */} },
    "round3": { "drafts": [/* SectionDraft */] },
    "round4": { "polished": [/* PolishedSection */] },
    "round5": { "meta": {/* Metadata */}, "images": [/* ImagePrompt */] },
    "round6": { "coherence": {/* CoherenceReport */} },
    "round7": { "wp": {/* WPDraft */} }
  },
```

```
  "diagnostics": [{ "round": 3, "ms": 8123, "notes": "..." }]
}
```

`vectors/{id}`

```
{
  "kind": "post|section|title",
  "embedding": [0.012, ...],
  "ref": { "runId": "", "draftId": "", "sectionId": "" },
  "textHash": "sha256",
  "createdAt": "serverTimestamp"
}
```

`cache/serpapi/{hash}`

```
{ "query": "string", "region": "string", "payload": {/* raw */},
"expiresAt": 1699999999 }
```

---

5) Cloud Functions / Genkit Scaffolding (TypeScript)

Project structure:

```
functions/
  src/
    clients/
      serp.ts          # SerpApi + caching
      hf.ts            # HF text + embeddings wrapper
      rephrasy.ts      # Rephrasy.ai client
      wp.ts            # WordPress REST client
    rounds/
      r0_trends.ts
      r1_ideation.ts
      r2_outline.ts
      r3_drafting.ts
      r4_polish.ts
      r5_metadata.ts
      r6_coherence.ts
      r7_publish.ts
    orchestrator.ts    # state machine / dispatcher
    types.ts
  .env                 # local dev only
  package.json
  tsconfig.json
```

Example: **HF client**

```
// src/clients/hf.ts
import fetch from "node-fetch";
const HF = process.env.HF_TOKEN!;

export async function inferText(model: string, input: unknown, params:
Record<string,unknown> = {}) {
  const res = await fetch(`https://api-inference.huggingface.co/models/$
{model}`, {
    method: "POST",
    headers: { "Authorization": `Bearer ${HF}`, "Content-Type": "application/
json" },
    body: JSON.stringify({ inputs: input, parameters: params })
  });
  if (!res.ok) throw new Error(`HF ${model} ${res.status}`);
  return res.json();
}

export async function embedText(model: string, texts: string[]):
Promise<number[][]> {
  const res = await fetch(`https://api-inference.huggingface.co/pipeline/
feature-extraction/${model}`, {
    method: "POST",
    headers: { "Authorization": `Bearer ${HF}`, "Content-Type": "application/
json" },
    body: JSON.stringify({ inputs: texts, options: { wait_for_model:
true } })
  });
  if (!res.ok) throw new Error(`HF embed ${res.status}`);
  return await res.json();
}
```

Example: **Round 0**

```
// src/rounds/r0_trends.ts
import { getSerpTrends } from "../clients/serp";
import { db, FieldValue } from "../firebase";

export async function round0FetchTrends(runId: string, seeds: string[],
region = "in") {
  const trendItems = await getSerpTrends(seeds, region);
  await db.doc(`runs/${runId}`).update({
    round: 0,
    "artifacts.round0": { trendItems },
    updatedAt: FieldValue.serverTimestamp()
  });
```

```
  return trendItems;
}
```

Example: **Orchestrator** (simplified)

```
// src/orchestrator.ts
export async function runPipeline(params: { seeds: string[]; region?:
string; brandVoice?: string }) {
  const runId = createRun(params);
  const trends = await r0.round0FetchTrends(runId, params.seeds,
params.region);
  const ideas  = await r1.round1IdeateTopics(runId, trends);
  const outline= await r2.round2Outline(runId, ideas[0]);
  const drafts = await r3.round3DraftSections(runId, outline);
  const pol    = await r4.round4Polish(runId, drafts, params.brandVoice ??
"clear, helpful");
  const meta   = await r5.round5Metadata(runId, outline, pol);
  const coh    = await r6.round6Coherence(runId,
[outline.title, ...pol.map(p=>p.content)]);
  if (coh.overall >= 0.75 && (coh.duplicates?.length ?? 0) === 0) {
    await r7.round7Publish(runId, pol, meta.meta);
  }
  return { runId };
}
```

Genkit (optional) — define each round as a `flow()` and orchestrate via `run()` to gain tracing and partial re-runs.

---

6) Security & Configuration

- **Secrets**: Store in **Google Secret Manager**. Bind access to Cloud Functions SA only.
- `SERP_API_KEY`
- `HF_TOKEN`
- `REPHRASY_API_KEY`
- `WP_BASE_URL`, `WP_USERNAME`, `WP_APP_PASSWORD` (or JWT token)
- **Local Dev**: Use `.env` for emulator only; never commit.
- **IAM**: Principle of least privilege; separate deployer vs runtime service accounts.
- **Network**: If WP blocks unknown IPs, consider egress via Serverless VPC Access.

---

7) Rephrasy.ai Integration (Round 4 or Post-Processing)

- Call as a post-processor on polished sections to “humanize” while preserving citations.
- Provide a strict mask of allowed transformations (syntax, vocabulary, cadence). Reject if hallucination diff exceeds threshold.

Client sketch:

```ts
// src/clients/rephrasy.ts
export async function humanize(text: string, style: string) {
  const res = await fetch("https://api.rephrasy.ai/v1/rephrase",
{ /* headers with API key */ });
  // include `preserve_tokens: ["[1]","[2]"]` and `max_diff_ratio`
}
```

---

8) Embedding-Based Coherence & Dedup (Round 6)

- Model: `sentence-transformers/all-MiniLM-L6-v2` (fast, cheap) initially.
- Compute vectors for: title, each polished section, full post.
- **Coherence**: `mean(cos(section, full_post))` target ≥ 0.75; flag low sections for rewrite.
- **Dup check**: compare `full_post` vs last 500 vectors; if cosine ≥ 0.92, mark as duplicate and skip publish.
- **Storage**: Save vectors in `vectors/*` with SHA256 of text. For small scale, Firestore scan is acceptable; upgrade later to a dedicated vector DB.

---

9) WordPress Publishing (Round 7)

- Use WordPress **Application Passwords** or JWT plugin. Prefer Application Passwords for ease + revocation.
- Endpoint: `POST /wp-json/wp/v2/posts` with `status: "draft"`.
- HTML assembly: preserve headings and add a **References** section using the collected placeholders.
- Attach featured image later when image generation is integrated.

---

10) Cost-Lean Defaults & Upgrades

- **TinyLlama / Gemma 2–7B**: HF Inference API free tier (cold starts acceptable).
- **Phi-3 Mini, Mistral Small, LLaMA 3.1 8B**: choose hosted endpoints with free credits or switch to nearest small alternatives if constrained.
- Embeddings: `all-MiniLM-L6-v2` via HF; later upgrade to `e5-base` and ANN index.
- Caching: 24–48h TTL for SerpApi responses to reduce spend.

---

11) Testing & Observability

- Use **Firebase Emulator Suite** for Firestore + Functions.
- Snapshot tests per round: fixed inputs → deterministic outputs via mocked HF responses.
- Structured logs with `round`, `ms`, `tokens` (if provider exposes usage).
- Retry policy: exponential backoff; circuit breaker on 5xx model errors.

---

12) Next Steps Checklist

1. Initialize Firebase project (`firebase init functions` with TypeScript).
2. Create Secret Manager entries and bind to Functions.
3. Scaffold clients and rounds as per structure above.
4. Implement Round 0–2 end-to-end with mocks.
5. Add Round 6 embeddings + dedup, then Round 7 WordPress draft posting.
6. Integrate Rephrasy.ai as optional post-processor.
7. Paste your Colab code here for precise mapping and removals.