AlumnAI

A BUILD GUIDE FROM ALUMNAI

The Automated Feed

Building a Self-Publishing Instagram Channel

A complete, working end-to-end system that finds the day's most important update on any topic you choose, writes it up, generates the artwork, and publishes it to Instagram — unattended, on a schedule, every day.

CONTENTS

What's inside

Ten sections, one working pipeline — built once, pointed at any topic you choose. Follow them in order; each one builds on the last.

REVISION NOTE — V5

This edition corrects three issues in Buffer's `createPost` mutation that only surfaced against Buffer's live schema — a wrong variable type, two undocumented required fields, and a scheduling mode that silently queues instead of publishing — fixes a headline-overflow bug in the artwork step, and documents the review-gate implementation concretely (Supabase schema, second workflow) rather than only conceptually. See Sections 07, 08, and 09.

BEFORE YOU START

This guide assumes basic comfort with Git, environment variables, and reading JSON. You don't need to be a developer — but you do need to be willing to follow instructions precisely. Every code sample is complete and working, not pseudocode.

SECTION 01

Why build this

A daily, on-topic Instagram presence — on any subject, without a daily manual task

> *The problem is rarely finding something to post about. It's finding the time to do it the same way, at the same standard, every single day — whatever the topic.*

An account that posts inconsistently loses the algorithm's attention and its audience's trust. An account that posts daily, at a reliable time, with a consistent visual identity, builds both — but only if someone is willing to research, write, design, and publish a post every morning, indefinitely.

This pipeline is topic-agnostic by design. Point it at AI news, and it tracks AI news. Point the same code at a running news story — a public inquiry, a conflict, an election — or at something lighter, like the most talked-about new gadget each week, and it does that instead. The only thing that changes between use cases is a handful of configuration lines: which sources it reads and what tone it writes in. Everything downstream is identical.

This guide builds a system that does that job for you. Once it's running, your daily involvement is optional: a 30-second glance at a draft before it goes out, if you choose to keep that gate in place (we recommend you do — more on that in Section 09).

5	<£15	0	1
PIPELINE STAGES	TYPICAL MONTHLY COST	MANUAL STEPS ONCE LIVE	CRON JOB DOING IT ALL

WHO THIS IS FOR

- **Solo builders and small brands** who want a credible, consistent presence on a topic they know well — without hiring a social media manager.
- **Anyone already comfortable with a GitHub repository** — you'll be editing YAML and environment variables, not writing an app from scratch.
- **People who want to own the pipeline** rather than rent it from a subscription tool. Every piece of this runs on infrastructure you control.

PICK YOUR TOPIC

The examples throughout this guide use an AI-news channel, because it's easy to follow. But the pipeline works identically for a running news story you're tracking closely — a public inquiry, a conflict, an election — or something entirely different, like the week's most talked-about new gadget. Section 05 shows exactly where the topic gets defined.

SECTION 02

The architecture

Five stages, one trigger, no human in the loop unless you want one

1. **Trigger**
   A scheduled GitHub Actions workflow fires at a fixed time each day (UTC-based cron). This is the only "always-on" part of the system — everything downstream only runs because this fired.

2. **Source**
   The workflow pulls the last 24 hours of headlines from a fixed set of RSS feeds relevant to your topic. No paid news API required at this volume.

3. **Select & write**
   The headline list is sent to the Claude API in a single call. Claude picks the most significant, shareable story and returns a structured JSON object: headline, angle, caption, hashtags.

4. **Generate**
   The story summary becomes an image prompt sent to FLUX 1.1 Pro (via fal.ai). The caption text is composited on top separately and wrapped to fit the frame, so typography stays crisp and never runs off the edge.

5. **Publish**
   The finished image and caption are sent to Buffer's API as an immediate share (`mode: shareNow`) — not queued for Buffer's own posting schedule. Buffer handles the actual Instagram connection through its own UI when you set up the channel; your pipeline never talks to Meta directly.

DESIGN PRINCIPLE: NO UNNECESSARY HOPS

Every stage in this pipeline exists because nothing downstream can do that job. There's no image DAM hosting the media separately, no separate orchestration platform coordinating the steps. GitHub Actions is the trigger and the runner; Buffer is the publisher. Fewer moving parts means fewer places for the daily job to silently fail.

WHY BUFFER, NOT THE INSTAGRAM GRAPH API DIRECTLY

Meta's own Graph API is technically free and technically direct — but getting there means creating a Meta Developer account, registering an app, navigating Business Suite's account verification, and surviving a genuinely hostile onboarding flow that can eat the best part of a day even for an experienced developer. Buffer's API needs a personal API key generated in your account settings, full stop. It's a few pounds a month at most (often free at this volume), and it buys you out of the worst part of this entire build.

SECTION 03

The stack, and why

Every tool here was chosen against real alternatives — not by default

LAYER	TOOL	WHY THIS, NOT THE ALTERNATIVE
Orchestration	GitHub Actions	Free, version-controlled, and you're already running code. A visual workflow tool (n8n, Zapier) adds overhead for one linear daily job.
News source	RSS feeds	Free and structured. A paid news API buys you nothing extra at one post a day.
Selection & copy	Claude API	One call does selection and writing together — splitting it into two calls adds cost and latency with no quality gain at this scale.
Image generation	FLUX 1.1 Pro (fal.ai)	Midjourney has no API — disqualified outright for automation. FLUX gives the best quality-to-cost ratio for unattended, API-driven generation.
Text overlay	Template render, not baked-in AI text	FLUX (and most image models) still render typography inconsistently. Compositing the caption separately — with proper word-wrap — keeps it crisp and fully on-frame every time.
Publishing	Buffer API	A personal API key from your account settings — no Meta developer app, no business verification, no OAuth registration. The alternative is going head-to-head with Meta's own developer console, which is worth avoiding at almost any cost.
Hosting the media URL	Supabase Storage	Buffer needs a public URL to fetch media from, same as any publishing API. Reusing infrastructure you already run beats adding a dedicated DAM like Cloudinary for one job.
Approval queue	Supabase table (Postgres)	The review gate needs somewhere durable to hold a draft between "generated" and "approved." A single table with a status column does the job without a bespoke admin app.

WHAT WE DELIBERATELY LEFT OUT

The Instagram Graph API, direct — technically free, but the setup cost is a Meta Developer account, an app registration, and Business Suite's verification maze. Buffer's API trades a few pounds a month (often nothing, at this volume) for skipping all of it.

Cloudinary — a full DAM platform to do the job of "give me a public URL" is overkill unless you're already using its transformation features elsewhere in your stack.

> **THE ONE-LINE VERSION**
>
> Use the tool that's already free, already running, or already yours — and only add a paid layer when it does something nothing else in the stack can. Here, that paid layer is Buffer, and what it buys you is never having to speak to Meta's developer console directly.

SECTION 04

Foundations

Set these up once, before writing any pipeline code

ACCOUNTS YOU'LL NEED

- **A Buffer account** (the free plan is enough for one channel posting once a day) with your Instagram professional account connected — done entirely through Buffer's normal "Connect a channel" flow, the same login screen you'd use by hand.
- **An Anthropic API key** for the Claude API calls (selection, writing).
- **A fal.ai account** (or Replicate) for FLUX image generation, billed pay-per-image.
- **A Supabase project** — you likely already have one — with a Storage bucket set to public read, and the `posts` table from Section 09.
- **A GitHub repository**, private is fine, to hold the workflow and pipeline scripts.

REPOSITORY SECRETS

Store every credential as a GitHub Actions repository secret — never commit a key to the repo, even a private one.

```
Settings > Secrets and variables > Actions > New repository secret

ANTHROPIC_API_KEY        sk-ant-...
FAL_KEY                  fal_...
BUFFER_API_KEY           personal key, see below
BUFFER_CHANNEL_ID        your Instagram channel's ID in Buffer
SUPABASE_URL             https://xxxx.supabase.co
SUPABASE_SERVICE_KEY     service_role key, not anon
```

REPOSITORY STRUCTURE

```
alumnai-ai-feed/
├─ .github/
│   └─ workflows/
│        ├─ daily-post.yml         # the cron trigger, Section 09
│        └─ publish-approved.yml   # the approve-and-publish half, Section 09
├─ scripts/
│   ├─ fetch_updates.py           # Section 05
│   ├─ select_and_write.py        # Section 06
│   ├─ generate_image.py          # Section 07
│   ├─ publish_buffer.py          # Section 08
│   ├─ review_gate.py             # Section 09
│   ├─ run_pipeline.py            # orchestrates the above
│   └─ publish_approved.py        # Section 09
└─ requirements.txt
```

GETTING YOUR BUFFER API KEY AND CHANNEL ID

In Buffer, go to **Settings → API** (or visit `publish.buffer.com/settings/api` directly) and generate a personal API key — no app registration, no callback URL, no approval wait. It doesn't expire on a fixed schedule the way OAuth tokens do, though you can regenerate it any time if it's ever exposed. Section 08 shows a one-line query to fetch your channel ID once the key is in hand.

SECTION 05

Sourcing your topic

Pull the last 24 hours from a fixed set of feeds — whatever the subject

Pick 3-5 feeds that consistently cover your chosen topic well, and filter every run to items published in the last 24 hours. Fewer, better feeds beat a large noisy list — you want Claude choosing between five strong candidates, not fifty mediocre ones. This is the only part of the pipeline that changes meaningfully between subjects; everything from Section 06 onward is identical regardless of what `TOPIC` and `FEEDS` are set to.

TOPIC	FEED EXAMPLES
AI news	TechCrunch AI, The Verge AI, Anthropic & OpenAI blogs
A public inquiry (e.g. grooming gangs inquiry)	BBC UK, The Guardian UK, Sky News, official inquiry press releases
A conflict (e.g. Iran)	Reuters World, AP News, BBC World, Al Jazeera
Consumer tech / gadgets	The Verge, Engadget, Wired Gear, r/gadgets RSS

```python
# scripts/fetch_updates.py
import feedparser
from datetime import datetime, timedelta, timezone

# Set these two values per channel — everything else in the
# pipeline reads from them and needs no further changes.
TOPIC = "AI news"   # e.g. "the Iran conflict", "new gadget releases"

FEEDS = [
    "https://techcrunch.com/category/artificial-intelligence/feed/",
    "https://www.theverge.com/rss/ai-artificial-intelligence/index.xml",
    "https://www.anthropic.com/news/rss.xml",
    "https://openai.com/blog/rss.xml",
]

def fetch_recent(hours=24):
    cutoff = datetime.now(timezone.utc) - timedelta(hours=hours)
    items = []
    for url in FEEDS:
        feed = feedparser.parse(url)
        for entry in feed.entries:
            published = datetime(*entry.published_parsed[:6], tzinfo=timezone.utc)
            if published >= cutoff:
                items.append({
                    "title": entry.title,
                    "summary": entry.get("summary", ""),
                    "link": entry.link,
                    "source": feed.feed.get("title", url),
                    "published": published.isoformat(),
                })
    return items

if __name__ == "__main__":
    import json
    print(json.dumps(fetch_recent(), indent=2))
```

FALLBACK FOR A QUIET DAY

If no feed items land in the 24-hour window — it happens on any topic, even a fast-moving one — have `fetch_updates.py` widen to 48 hours before giving up. Better to occasionally repeat a strong story's angle than to skip a scheduled post entirely and break your posting streak.

SECTION 06

Selecting & writing

One Claude API call: pick the story, write the copy, return structured JSON

Send the day's headline list to Claude in a single request, with strict instructions to return only JSON. This is the one call doing the most work in the whole pipeline — it's worth spending time getting the system prompt right. The `TOPIC` variable from Section 05 drops straight into the prompt, so the same script serves every channel.

```python
# scripts/select_and_write.py
import anthropic, json
from fetch_updates import TOPIC

client = anthropic.Anthropic()

SYSTEM = f"""You are the voice of an Instagram account covering {TOPIC}.
Pick the single most significant, shareable update from the list.
Write a caption in a punchy, plain-English tone — no hype,
no emoji overload, 2-3 short paragraphs, ending with a question
to drive comments. Include 8-10 relevant hashtags.
Return ONLY valid JSON, no preamble, no markdown fences, matching:
{{"headline": str, "source_url": str, "image_brief": str,
 "caption": str, "hashtags": [str]}}"""

def select_and_write(news_items):
    response = client.messages.create(
        model="claude-sonnet-4-6",
        max_tokens=1000,
        system=SYSTEM,
        messages=[{"role": "user",
                   "content": json.dumps(news_items)}]
    )
    return json.loads(response.content[0].text)
```

THE IMAGE_BRIEF FIELD

Ask Claude to also produce a short, concrete visual description of the story — this becomes the FLUX prompt in the next stage, written in the same call so the visual and the copy stay conceptually aligned rather than generated in isolation from each other.

GUARDING AGAINST MALFORMED OUTPUT

Wrap the `json.loads()` call in a try/except and, on failure, retry once with an explicit "your last response wasn't valid JSON" follow-up message. If it fails twice, have the workflow exit cleanly and notify you rather than publish a broken post.

TONE MATTERS MORE ON SERIOUS TOPICS

A gadget channel can afford a light, punchy voice by default. A channel tracking a live conflict or a public inquiry can't — extend the system prompt with explicit tone constraints (measured, sourced, no speculation beyond what the article states) and keep the review gate from Section 09 firmly switched on. The cost of an AI misjudging tone is much higher on a sensitive story than on a product launch.

SECTION 07

Generating the artwork

FLUX for the base image, a template pass for crisp — and fully on-frame — text

A Generate the base image with FLUX

Send the `image_brief` from Section 06 to FLUX 1.1 Pro via fal.ai, requesting a portrait aspect ratio suited to Instagram's best-performing feed format.

```python
# scripts/generate_image.py
import fal_client

def generate_base_image(image_brief):
    result = fal_client.subscribe(
        "fal-ai/flux-pro/v1.1",
        arguments={
            "prompt": image_brief,
            "image_size": "portrait_4_3",
            "num_images": 1,
        },
    )
    return result["images"][0]["url"]
```

B Composite headline text on top, wrapped to fit

Don't ask FLUX to render text in-image — typography is still the weak point of every image model. Instead, download the base image and overlay the headline with Pillow, measuring it against the frame width first so it wraps onto multiple lines rather than running off the edge.

```python
# continued: scripts/generate_image.py
from PIL import Image, ImageDraw, ImageFont

def _wrap_text(draw, text, font, max_width):
    words, lines, current = text.split(), [], ""
    for word in words:
        candidate = f"{current} {word}".strip()
        if draw.textlength(candidate, font=font) <= max_width:
            current = candidate
        else:
            lines.append(current)
            current = word
    lines.append(current)
    return lines

def composite_post(base_image_path, headline, out_path):
    img = Image.open(base_image_path).convert("RGB")
    draw = ImageDraw.Draw(img)
    font = ImageFont.truetype("assets/PlayfairDisplay-Bold.ttf", 64)
    margin = 60

    lines = _wrap_text(draw, headline, font, img.width - 2 * margin)
    wrapped = "\n".join(lines)
    text_h = draw.multiline_textbbox((0, 0), wrapped, font=font, spacing=16)[3]
    band_height = text_h + 2 * margin

    # dark band behind text, sized to the wrapped line count
    overlay = Image.new("RGBA", img.size, (0, 0, 0, 0))
    band = ImageDraw.Draw(overlay)
    band.rectangle([0, img.height - band_height, img.width, img.height],
                   fill=(10, 26, 47, 210))
    img = Image.alpha_composite(img.convert("RGBA"), overlay)

    draw = ImageDraw.Draw(img)
    draw.multiline_text((margin, img.height - band_height + margin), wrapped,
                        font=font, fill="white", spacing=16)
    img.convert("RGB").save(out_path, quality=92)
```

WHY THIS CHANGED

The original single-line `draw.text()` call had no width check, so any headline longer than roughly 20 characters ran straight off the right edge of the frame — invisible in testing, obvious the moment a real headline hit it on a live post. Measuring and wrapping first, then sizing the legibility band to the wrapped text, fixes it for headlines of any length.

SECTION 08

Publishing via Buffer

One GraphQL call, corrected against Buffer's actual schema

Buffer's API is GraphQL, hit with a single endpoint (`https://api.buffer.com`) and your personal API key from Section 04 sent as a Bearer token. Before your first post, run a one-off query to find your channel's ID — you'll only need to do this once and can hardcode the result as `BUFFER_CHANNEL_ID`.

```
# one-off: find your Instagram channel ID
import requests, os
TOKEN = os.environ["BUFFER_API_KEY"]
query = """
query GetChannels {
  account { organizations { id
      channels { id serviceType displayName }
  } }
}
"""
r = requests.post("https://api.buffer.com",
    headers={"Authorization": f"Bearer {TOKEN}"},
    json={"query": query})
print(r.json())  # find the entry with serviceType "instagram"
```

With the channel ID in hand, publishing is a single mutation. The image URL must be publicly reachable — the Supabase Storage URL from Section 07 works directly.

```
# scripts/publish_buffer.py
import requests, os
TOKEN = os.environ["BUFFER_API_KEY"]
CHANNEL_ID = os.environ["BUFFER_CHANNEL_ID"]
MUTATION = """
mutation CreatePost($text: String!, $channelId: ChannelId!,
    $imageUrl: String!, $postType: PostType!, $shouldShareToFeed: Boolean!) {
  createPost(input: {
    text: $text
    channelId: $channelId
    schedulingType: automatic
    mode: shareNow
    assets: [{ image: { url: $imageUrl } }]
    metadata: { instagram: { type: $postType, shouldShareToFeed: $shouldShareToFeed } }
  }) {
    ... on PostActionSuccess { post { id } }
    ... on MutationError { message }
  }
}
"""
def publish_post(image_url, caption):
    r = requests.post("https://api.buffer.com",
        headers={
            "Content-Type": "application/json",
            "Authorization": f"Bearer {TOKEN}",
        },
        json={
            "query": MUTATION,
            "variables": {
                "text": caption,
                "channelId": CHANNEL_ID,
                "imageUrl": image_url,
                "postType": "post",
                "shouldShareToFeed": True,
            },
        })
    body = r.json()
    if body.get("errors"):
        raise RuntimeError(f"Buffer API error: {body['errors']}")
    result = body["data"]["createPost"]
    if "message" in result:
        raise RuntimeError(f"Buffer rejected the post: {result['message']}")
    return result["post"]["id"]
```

THREE CORRECTIONS AGAINST BUFFER'S LIVE SCHEMA

Type: `channelId` is the custom `ChannelId` scalar, not `String` — a mismatch Buffer's validator rejects outright.
Required Instagram metadata: every Instagram post needs `metadata.instagram.type` (`post`/`reel`/`story`/etc.) and `metadata.instagram.shouldShareToFeed` — omit either and the mutation fails validation before it reaches Instagram at all.
Mode: `mode: addToQueue` adds the post to Buffer's own queue for its next scheduled slot — it does not publish. Use `mode: shareNow` if "approved" is meant to mean "goes out now."

RATE LIMITS

Buffer's free plan allows 100 API requests per 24 hours and 3,000 per 30 days — not remotely a concern at one post a day. If you later add multiple channels or higher-frequency posting, paid plans raise this ceiling.

SECTION 09

Scheduling & the review gate

Two workflows: one drafts and waits, the other publishes what you approve

```yaml
# .github/workflows/daily-post.yml
name: Daily Topic Post
on:
  schedule:
    # 08:00 UK time = 07:00 UTC (or 08:00 UTC in winter)
    - cron: '0 7 * * *'
  workflow_dispatch: {}    # lets you trigger it manually to test
jobs:
  post:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with: {python-version: '3.11'}
      - run: pip install -r requirements.txt
      - run: python scripts/run_pipeline.py
        env:
          ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
          FAL_KEY: ${{ secrets.FAL_KEY }}
          BUFFER_API_KEY: ${{ secrets.BUFFER_API_KEY }}
          BUFFER_CHANNEL_ID: ${{ secrets.BUFFER_CHANNEL_ID }}
          SUPABASE_URL: ${{ secrets.SUPABASE_URL }}
          SUPABASE_SERVICE_KEY: ${{ secrets.SUPABASE_SERVICE_KEY }}
          REVIEW_GATE: ${{ vars.REVIEW_GATE || 'true' }}
```

THE QUEUE, CONCRETELY

With `REVIEW_GATE` true (the default), `run_pipeline.py` writes the finished draft to a Supabase table instead of calling `publish_post()` directly. That table is the review gate:

```sql
-- run once in the Supabase SQL editor
create table posts (
  id uuid primary key default gen_random_uuid(),
  created_at timestamptz not null default now(),
  topic text not null,
  headline text not null,
  source_url text,
  caption text not null,
  hashtags text[] not null default '{}',
  image_url text not null,
  status text not null default 'pending'
    check (status in ('pending','approved','rejected','published','failed')),
  published_at timestamptz,
  buffer_post_id text,
  error text
);
```

A second workflow, `publish-approved.yml`, runs independently on a 30-minute cron (and on demand): it selects every row with `status = 'approved'`, calls `publish_post()` for each, and updates the row to `published` with the returned Buffer post id — or to `failed` with the error recorded, if Buffer rejects it. Approving means flipping one row's status in the Supabase table editor; nothing more is needed to trigger a publish.

> **WHY THE GATE MATTERS**
>
> An LLM confidently misreading a headline, or FLUX generating something off-brand, is rare but not impossible. A 30-second daily glance costs you almost nothing and catches the one post a year you'd otherwise regret publishing unattended. Set the `REVIEW_GATE` repository variable to `false` once you trust the pipeline, and `daily-post.yml` will call `publish_post()` directly with no draft step.

SECTION 10

Running costs & troubleshooting

What it actually costs monthly, and the failure modes to expect

ITEM	TYPICAL MONTHLY COST
GitHub Actions (public runners, ~2 jobs/day)	Free — well within the free-tier minutes
Claude API (one call/day, small prompt)	Under £1
FLUX via fal.ai (one image/day)	£1–3
Supabase Storage & table	Free tier covers this easily
Buffer (one channel, free plan)	Free — upgrade only if you add more channels or higher-frequency posting
Total	**Roughly £2–5 a month**

COMMON FAILURE POINTS

✓ **Buffer key revoked or regenerated.** If you ever regenerate the API key in Buffer's settings, the old one stops working immediately. Update the GitHub secret straight away.

✓ **Malformed JSON from Claude.** Rare, but handle it — see Section 06's retry pattern.

✓ **channelId typed as String.** Buffer's validator rejects it outright with `GRAPHQL_VALIDATION_FAILED` — it must be the `ChannelId` scalar.

✓ **Missing Instagram metadata.** `metadata.instagram.type` and `.shouldShareToFeed` are both required on every Instagram post; the mutation fails validation without them.

✓ **Post accepted but nothing appears on Instagram.** Almost always `mode: addToQueue` — it queues for Buffer's own schedule rather than publishing. Use `shareNow`.

✓ **`MutationError` from Buffer.** Usually an image URL Buffer couldn't fetch — check the Supabase bucket is genuinely public and not behind a signed/expiring link.

✓ **No news in the 24h window.** Handled by the 48h fallback in Section 05.

WHERE TO TAKE THIS NEXT

Once the daily post is running reliably, the natural extensions are: a second channel on an entirely different topic — the same codebase, a different `TOPIC` and `FEEDS` block and a second Instagram account — a weekly roundup carousel, automatic performance logging back into Supabase so you can see which stories and image styles perform best, and eventually Reels — which trades the FLUX step for a short AI-video generation call instead.

WANT THIS BUILT FOR YOU?

AlumnAI builds and hands over working pipelines like this one — configured for your topic, your brand, and your account, ready to switch on. Whether it's AI news, a story you're following closely, or a niche you want to own on Instagram, get in touch via **alumnai.co.uk**.