6-Prompts

Note: All specific references to the company that deploys these prompts have been replaced with a fake, stand-in company called JobBuilder.com.

Refurbishment Project Guidelines

Use Current Date: Infer from system time (e.g., [Month Year] like October 2025) for all date-sensitive queries.

Project Task

Automate 2023 Job Builder article updates: Verify entities; fact-check citation-worthy claims with current data/sources; add high-value info; improve SEO/links; assemble preserved final with clean citations. Minimize changes to retain structure/tone/voice/text.

- **Input:** Raw pasted JB text (messy OK). Extract main (title/intro/H2s/body); ignore nav/ads. Clean artifacts internally; preserve original in outputs.
- **Outputs Chain:** P1 (verification) → P2 (expansions) → P3 (intro/Working Article) → P4 (SEO) → P5 (citations/sources) → P6 (HTML final).
    - **Multi-Subtask:** Use ### Steps. All prompts ref these Guidelines.
- **Cross-Runs:** Handles 100s of JB articles. Editor suggestions: Use neutral examples.

Preservation

Preserve original structure, tone, voice, and text as much as possible. Match voice: Conversational, repetitive (e.g., 'rigorous training'), casual quirks (e.g., 'you too can land a job').

- **For Updates/Additions:** replace only exact outdated elements verbatim without rewriting, condensing, or altering sentence structure/context.
- Adhere to **Theseus's Ship Rule** for minimal edits to retain original character.
- Transitions: 1-5 words for rhythm; ^n after punctuation (treat as formatting).
    - Maintain original voice in added transitions (e.g., 'that said' for contrast, 'in other words' for rephrase — use singly, not chained; ensure it fits context without redundancy); mimic flow/enhance relatability.
- **Growth/Shrink**: ≤20% growth (≤10% filler/section); ≥95% retention (≤7% condense). Audit in prompts.

Verification Rules

- **Chain of Verification:** ≥93% confidence (HTTP 200/snippet) via ≤3 chains (official → news/gov → waybaJB); ≥2 methods; self-critique (<93% refine); flag >3 manual. Hierarchy: Primary (official/.gov) > Secondary (Glassdoor/Statista/news) > Tertiary (CourseReport).

- **Salaries:** BLS first; 1-2 secondaries (e.g., '[role] salary [Date] site:glassdoor.com').
- **Tools:** code_execution (audits/regex/wc); web_search_with_snippets (SEO/trends); x_semantic_search (additions, topic-adapted). Parallel OK; ≤3 chains on fails/paywalls; X social-only.
- **Inclusion: ≥1 verified source (URL/snippet);** no source = omit/'recent data'. No fabrication; evergreen rare.
- **Fallback:** code/token fail (>80% est.) → auto-chunk [text[i:i+5000] for i in range(0,len(text),5000)]; log '[FALLBACK: Chunked X]'.
- **Priority:** Accuracy > speed; append [Date] to queries.

Key Definitions

- **Citation-Worthy:** Any claim a reasonable skeptic would need proof for, i.e., quantifiable skeptics need: (stats/costs/salaries/growth/rankings/durations/regulations/quotes/program lists/scholarship details). Ex: '800 hours', '$5K award', '8% growth'. Include lists (e.g., VET TEC).
- **Non-Citation worthy:** Broad (e.g., 'prices vary'); skip narrative/opinions/marketing.
- **High-Value:** Verified citation-worthy + relevant (trends/AI/alternatives) for job-change help; no filler.
- **Entities:** Schools/bootcamps/programs/courses/resources/careers/degrees; treat specifics separate (e.g., 'Full Stack' under school).
- **Echo:** Exact quantifiable match (e.g., '$105/credit' body/table); verify once P1; dual ^n in P5.
- **Original Article:** Full pasted text (P1 input); Theseus's Ship core.
- **Breakout:** H2/H3 dive post-table/list (e.g., programs table → overviews).
- **Placement:** P1 Step 2/P5 JSON: {'claim': '$169', 'locations': ['para 1', 'row 2'], 'n': 14}.

Editor Approval

Proposals (P1 deletions/cites, P2 adds, P3 intro/KTs, P4 SEO, P5 sources) need 'approve all' or 'reject IDs X,Y,Z'. Batch/zero-input: Auto-approve ≤3 flags (drop <5% growth); log '[AUTO-APPROVED: Minors dropped]'. Enable 'batch=true' in paste; fallback manual.

Output Formatting

For P1-5: Markdown (bullets -, tables | Col |, ## H2/### H3); **for P6:** HTML code block (Google Docs paste).

- **General:** No narrative/meta; plain text; en dash – (no —). Snippets ≤20 words/…
- **Token Handling:** code_execution est. (len(text.split())*4); chunk >80% limit (3-5 parts/H2); reassemble artifacts. Prioritize summaries.
- **Chunk Snippet (reuse):**

```

import re

if len(text) > 5000:

    chunks = re.split(r'(?<=</h2>|\n\n)', text)[:6]  # H2 ends, cap 6 for fuller scan

    full_scan = re.findall(claims_regex, '\n\n'.join(chunks))  # Cross-chunk claims

    if len(full_scan) > len(re.findall(claims_regex, text[:5000])):  # If more found, use full

        full = '\n\n'.join(chunks)

    else:

        full = text[:5000]  # Fallback

    print(f'[CHUNKED: {len(chunks)} parts; extra claims: {len(full_scan) -
len(re.findall(claims_regex, text[:5000]))}]')

else:

    full = text
```

Artifact Summaries: All P1-6 end JSON summary (≤250 tokens): entities/claims/sources/adds/sections/wc_audit/flags/for_next. P2-5: Load via code (json.loads('recall from history')). Full Recovery: If 'full_available', code: report_text = 'prior Markdown'; full_claims = re.findall(r'\| ID \| ... \|', report_text); print(json.dumps({'full_claims': full_claims})).

###These apply to all upcoming prompts in this series. Do you understand and confirm?###

1 - Verification & Fact-Check

###Job Builder Article Below###

###Prompt 1 — Verification & Fact-Check###

Task

Verify programs/schools/services/resources in pasted 2023 article text. Update citation-worthy claims (stats/costs/salaries/rankings/regulations) with current sources. **Output**: Verification Table, Deletion Report, Citations Report + Artifact Summary.

Preservation

See ###Refurbishment Project Guidelines###.

Prompt 1 Tools/Notes

- **Status:** Parallel browse_page official + web_search_with_snippets '[entity] status [Date] site:news OR reddit.com' (≤2 tools/call; prioritize browse HTTP 200).
- **Claims:** Primary official/BLS; secondary BLS/Statista (≤2/claim). ≥1 primary; borderline (e.g., 'affordable' implying $X) → chain 1x, else non-citation.
- **Parallel:** Merge results in reports (e.g., "Source: [URL] + [snippet]").

Task Sequence

###Step 0: Detect Topic

Run code_execution for topic, entities/claims extraction, echoes, highlights, placements, flags. (Code handles all):

```
import re, json
try:
    recall_summary = 'recall from history: Article summary like {"title": "[title]", "body_snippet": "[first 100 words]"}'
    text = re.findall(r'"body_snippet":\s*"([^"]+)"', recall_summary)[0] if re.search(r'"body_snippet"', recall_summary) else input('Paste article text: ')
    title = re.findall(r'"title":\s*"([^"]+)"', recall_summary)[0] if re.search(r'"title"', recall_summary) else 'Sample Title'
    wc = len(text.split())
```

```
    long_article = wc > 5000
    if long_article:
        chunks = re.split(r'(?<=</h2>|\n\n)', text)[:6]
        full_text = '\n\n'.join(chunks)
        extra_claims = len(re.findall(claims_regex, full_text)) - len(re.findall(claims_regex,
text[:5000]))
        print(f'[CHUNKED: {len(chunks)} parts; extra claims: {extra_claims}]')
    else:
        full_text = text
    combined = (title + ' ' + full_text).lower()
    topic_matches =
re.findall(r'bootcamp|career|financing|school|degree|veteran|dental|job|finance', combined)
    topic = topic_matches[0] if topic_matches else 'general'
    if topic == 'general':
        body_matches = re.findall(r'job|degree|career|finance|bootcamp|school', full_text.lower())
        topic = body_matches[0] if body_matches else 'general'
    if topic in ['veteran', 'dental', 'career']:
        entities_regex = r'\b[A-Z][a-z]+
(Bootcamp|School|Program|Degree|Financing|Career|Veteran|Hygiene|Job)\b'
    else:
        entities_regex = r'\b[A-Z][a-z]+ (Bootcamp|School|Program|Degree|Financing|Career)\b'
    entities = re.findall(entities_regex, full_text)
    claims_regex = r'\$[\d,]+\.?\d*(?:/credit| /semester| /year|\s*per\s*\w+)?|\d+%
(?:growth|grad|acceptance|licensure|pass|rate|discount)|\d+ (?:jobs?|openings|new
jobs?|hours?|credits?|semesters?|years?|enrollments?|students?|awards?)|ranked
#\d+|#\d+|GPA \d+\.\d+|\d+-\d+ (?:credits?|hours?|jobs?|tuition|salary|costs?)|average
(?:\$[\d,]+\.?\d*|\d+%|\d+ (?:jobs?|hours?|credits?))|costs range from \$[\d,]+ to \$[\d,]+|most
affordable (?:\$[\d,]+|\d+ credits?)|(?:as low as|projected|estimated) (?:\$[\d,]+|\d+%|\d+
(?:credits?|hours?))'
    claims = re.findall(claims_regex, full_text)
    table_regex = r'\|.*?[\n\|]+.*?(?:\n|$)'
    tables = re.findall(table_regex, full_text, re.DOTALL)
    for table in tables:
        rows = [row.strip() for row in table.split('\n') if row.strip() and '|' in row]
        for row in rows:
            if not re.match(r'^\|[\s-]*\|', row):
                table_entities = re.findall(entities_regex, row)
                table_claims = re.findall(claims_regex, row)
                entities.extend(table_entities)
                claims.extend(table_claims)
    entities = list(set(entities))
    claims = list(set(claims))
    echo_dict = {c: full_text.count(c) for c in claims if full_text.count(c) > 1}
    flags = [f"Echo: {k} in {v} locations" for k, v in echo_dict.items() if v > 1]
```

```
    token_est = len(full_text.split()) * 5.2  # Adjusted for Markdown overhead
    print(f'[DEBUG: {len(entities)} entities, {len(claims)} claims detected]')
    highlights = []
    for i in range(min(6, len(entities) + len(claims))):
        entity = entities[i % len(entities)] if entities else 'Key program'
        claim = claims[i % len(claims)] if claims else 'verified details'
        highlights.append(f"- {entity}: {claim} (e.g., intro/H2 summary)")
    claim_placements = [{'claim': k, 'locations': [f'body/table {v} times'], 'n': i+1} for i, (k, v) in
enumerate(echo_dict.items())]
    cached = {'entities': entities, 'claims': claims}
    deletions = []  # Populated by Step 1 verification
    verified_entities = []  # Simulate Step 1 results
    for entity in entities:
        # Simulate: web_search_with_snippets(f'{entity} closed 2025 site:news OR reddit.com',
num=5)
        if 'Bootcamp' in entity and False:  # Replace with actual tool logic
            verified_entities.append({'entity': entity, 'status': 'ENDED'})
        else:
            verified_entities.append({'entity': entity, 'status': 'ACTIVE'})
    for v in verified_entities:
        if v['status'] == 'ENDED':
            full_text = re.sub(r'\b' + re.escape(v['entity']) + r'\b(?:\s*|\s*[^|]+)*', '', full_text)
            deletions.append(v['entity'])
            flags.append(f'[DELETED: {v["entity"]}]')
    print(f'[DEBUG: {len(deletions)} entities marked for deletion]')
    cached['deletions'] = deletions
    print(json.dumps({'detected_topic': topic, 'wc': wc, 'long_article': long_article, 'entities': entities,
'claims': claims, 'echo_dict': echo_dict, 'flags': flags, 'token_est': token_est, 'article_highlights':
highlights, 'claim_placements': claim_placements, 'cached_state': cached}))
except Exception as e:
    print(json.dumps({'detected_topic': 'general', 'wc': 0, 'long_article': False, 'entities': [], 'claims':
[], 'echo_dict': {}, 'flags': [f"Error: {str(e)}; fallback general"], 'token_est': 0, 'article_highlights': [],
'claim_placements': [], 'cached_state': {}}))

```

Step 1: Verify Entities

Extract/classify entities (schools/bootcamps/programs; specifics separate). Confirm activity.

- **Parallel**: browse_page official + web_search_with_snippets '[entity] status [Date] site:news OR reddit.com'.
- **Classify**: ACTIVE (enrollment/updates + confirm), ENDED (closure >18 months + news), UNVERIFIED (ambiguous; add to artifact {"status": "UNVERIFIED", "replacement_needed": true, "reason": "No [Date] data"}).

    - For ENDED, parallel web_search_with_snippets '[entity] closed 2025 site:news OR reddit.com' num=5; if 'closed' in ≥2 snippets and date >18 months ago, auto-delete entity from text via re.sub(r'\b' + re.escape(entity) + r'\b(?:\s*|\s*[^|]+)*', '', text); log '[DELETED: {entity}]' in Deletion Report.
- **Non-bootcamp:** Include 'Job|Degree|Career Path'; confirm site:official.edu/BLS.
- **ENDED**: Note replacement; flag P2. List for deletion; UNVERIFIED in Report; if ENDED >18 months (from web_search_with_snippets '[entity] closed date' num=3), auto-delete from body/tables via re.sub(entity_pattern, '', text) in P1 Step 0 code; log '[DELETED: {entity}]'.

###Step 2: Verify Claims

Process ACTIVE/UNVERIFIED; ENDED = 'No longer applicable'.

- **Cite All:** Every citation-worthy claim (even unchanged) → ^n via ≥1 primary (official/BLS; note 'evergreen' if timeless).
- **Chain of Verification**: Confirm/refresh; match 93% → reuse 'Verified current'; else verbatim update.
- **Borderline (scholarships/salaries):** Primary chain; propose ^n.
- **Echoes:** One ^n at first (body > table); propagate. Note 'Echo: [locations]' for P5; populate artifact['claim_placements'].
- **Updates:** Verbatim replace; ≤1-2 sources (primary first).
- **Tuition**: web_search_with_snippets '[school] [program] tuition [Date]'; careers: BLS/ed.gov chain; no results ≤2 → 'recent data'/'Unverifiable'. Non-US: Currency code_execution/web_search. Note aggregates ('Total incl. fees').
- **Attributes (accreditation):** browse_page '[entity] [attr] [Date]'; flag unsupported for P2 omit.
- **Lists**: Verify changes; multi-sources if complex.

Output Requirements (Steps 1-2)

Markdown:

- **Verification Table:** | Entity Name | Status | Source URL | Snippet (≤20 words) | Note (e.g., HTTP 200; replaced) |.
- **Deletion Report:** ENDED entities/unverified claims. Skip if none; tables/plain.
- **Citations Report:** | ID (#1+) | Location | Original Claim | Updated Claim | Source URL | Snippet | Echo: [locations] |. ENDED: 'No longer applicable'/N/A.| Replacement Needed: [yes/no, reason if yes].
    - All verified claims (updated/unchanged); Unverified: Flag P2 replacement.

Artifact Summary

JSON (≤250 tokens, ```json block): {"entities": [{"Ex Bootcamp": "ACTIVE"}, {"Iowa MS": "UNVERIFIED", "replacement_needed": true, "reason": "No 2025 data"}], "claims": [{"claim":

"$15K tuition", "source_url": "ex.com", "snippet": "2025 BLS", "status": "verified", "echo_locations": ["para 1"]}, ...], "deletions": [{"Old Prog": "ENDED"}], "unverified_replacements": [{"entity": "Iowa MS", "replacement_needed": true, "reason": "No 2025 data"}, ...], "flags": ["Propose alts for 2 UNVERIFIED"], "for_next": ["Use ACTIVE/claims/unverified_replacements for P2"], "article_highlights": ["- Costs: $15K BLS", ...], "full_available": true, "claim_placements": [full list]}. >10: {'count': N, 'examples': [:3], 'full_available': true}. (Keys for chain: unverified_replacements mandate P2 pitches.)

2 - Expansion Pitches

###Prompt 2 — Expansions###

Task

Use Original Article + P1 Artifact (recall entities/claims/sources). Propose high-value additions to sections + new sections for better education pathway resources. **Output**: Addition Report + Artifact Summary.

Preservation:

See ###Refurbishment Project Guidelines###.

Prompt 2 Tools/Notes

- **Pitches:** High-value citation-worthy; ≥1 primary source; no contradict/delete >5 words.
- **Parallel:** ≤2 tools/call (e.g., browse_page + web_search_with_snippets); merge in reports.
- **Load Fallback:** json.loads fail → {'fallback': true}; manual parse reports.
- **X/reddit/social media Posts:** Inspiration only (from_date='2024-01-01'); confirm ≥1 non-X primary.

Task Sequence

###Step 1: Pitch Additions to Existing Sections

Pitch from ≥1 primary (web_search_with_snippets '[topic] trends 2025 site:news OR statista.com', num=5; fallback x_semantic_search buzz, limit=5, min_score=0.2). No source ≤3 chains = no pitch. Match format/voice (duration/cost/length/subsections); seamless.

- Use tool results (e.g., web_search_with_snippets) to populate pitches list in code with relevant high-value additions, ensuring they match voice and are verified.
- **Re-Verify:** Post-pitch, re-verify with web_search_with_snippets 'addition claim Oct 2025 site:bls.gov OR ada.org OR official.edu' num=2; if <93% match (self-critique: exact $/% + date), revise pitch or omit; log '[UPDATED: {claim} to {new_source}]'.
- **Filter**: Self-critique ≥80% relevance; chain primaries if borderline; X inspiration only. Score >0.7 relevance; ≤12 additions (2/H2 + 2 new sections); snippets ≤75 words.
- **Improvements**: 1-2/H2; ≤50-word snippets (e.g., 'Blend after para 2: "AI boosts 20%...", etc.'); include source/snippet; high-value (BLS tips). Match P1 entity ID.
- **Replacements for Unverified**: Mandate 1-2 active alts per P1 'unverified_replacements' (load artifact['unverified_replacements']; web_search/browse '[entity] alternative 2025' ≥1 primary; match format/length; full draft for P3.
- **Lists/Tables**: Match size (e.g., 6→6); flag changes; tool-backed.

- **Breakouts**: Propagate table/list pitches to following H2/H3 (match entity ID; format/depth).

###Step 2: Pitch New Sections

1-2 H2/H3 (≤3 total, >0.7 relevance; est_growth ≤10%): 2-3 sentence summaries (placement "After H2 X"); BLS-prioritize (e.g., 'Tips: BLS 15% hires'). Precede FAQ.

- **Programs:** 1-3 active to lists (CourseReport → primary chain).
- **Additions:** 1-3 actionable (resume/interviews; BLS breakdowns); full 5-6 sentence drafts post-approval.
- **Breakout Prop:** Step 1 table pitch → matching H2/H3 overview (source/snippet; match format).

###Step 3: Internal Audit

Score relevance/snippet confirmation; flag growth. **Run code_execution:**

```
import json, re
try:
    try:
        artifact = json.loads('{"original_text": "default short text if not available"}')
    except json.JSONDecodeError:
        artifact = {"original_text": "default short text if not available"}
    recall_summary = 'recall from history: P1 summary like {"entities": {"count": 10, "examples": ["[example entity]"]}, "claims": {"count": 20}}'
    cached = artifact.get('cached_state', {})
    article_text = re.findall(r'"original_text":\s*"([^"]+)"', recall_summary)[0][:2000] if re.search(r'"original_text"', recall_summary) else artifact.get('original_text', 'default short text if not available')
    text_wc = len(article_text.split()) if article_text else 1000
    detected_topic = re.findall(r'bootcamp|career|financing|school|degree|veteran|dental|job|finance', article_text.lower())[0] if re.findall(r'bootcamp|career|financing|school|degree|veteran|dental|job|finance', article_text.lower()) else 'general'
    # Simulate tool: web_search_with_snippets(f'{detected_topic} trends 2025 site:news OR statista.com', num=5)
    # Parse results: Extract claims like '$X tuition' or 'X% growth' with source URLs
    pitches = [f"Add {detected_topic} trend: AI boosts 20% growth ^n (source: statista.com)", f"New {detected_topic} program: Online certification ^n (source: official.edu)"]
    list_regex = r'-\s.*?(?:\n-\s.*?)*|\|.*?(?:\|.*?\n)+'
    original_lists = re.findall(list_regex, article_text, re.DOTALL)
```

```
    list_sizes = [len(l.split('\n')) for l in original_lists if l]
    pitches_per_list = []
    for size in list_sizes:
        pitches_per_list.append(pitches[:size])
    pitches = [p for sublist in pitches_per_list for p in sublist]
    sections = artifact.get('sections', [{'content': article_text}])
    growth_per_section = []
    for sec in sections:
        sec_text = sec.get('content', '')
        sec_wc = len(sec_text.split())
        sec_pitches = [p for p in pitches if sec_text in p]
        sec_growth = (len(sec_pitches) * sum(len(p.split()) for p in sec_pitches) / len(sec_pitches) /
sec_wc * 100) if sec_pitches and sec_wc else 0
        growth_per_section.append(sec_growth)
    est_growth = max(growth_per_section) if growth_per_section else 0
    scores = [len(re.findall(r'BLS|source|trend|job-relevant', p)) * 2 for p in pitches]
    flags = ['Growth flag' if est_growth > 20 else 'OK']
    JB_topics = ['career', 'financing', 'degree', 'job', 'bootcamp', 'school']
    topic_boost = [s + 1 if any(term in p.lower() for term in JB_topics) else s for s, p in zip(scores,
pitches)]
    filtered_pitches = [p for i, p in enumerate(pitches) if topic_boost[i] > 1.4]
    unverified = artifact.get('unverified_replacements', [])
    for u in unverified[:2]:
        alt_pitch = f'Replace {u["entity"]} with New {detected_topic.capitalize()} Program ^n
(source: official.edu)'
        pitches.append(alt_pitch)
        flags.append(f'Replacement proposed for {u["entity"]}')
    cached['pitches'] = filtered_pitches + [p for p in pitches if 'Replacement' in p]
    print(json.dumps({'scores': dict(zip(pitches, topic_boost)), 'filtered_pitches': filtered_pitches,
'wc_audit': {'est_growth': est_growth, 'section_growth': growth_per_section}, 'flags': flags,
'cached_state': cached}))
except Exception as e:
    print(json.dumps({'scores': {}, 'filtered_pitches': [], 'wc_audit': {'est_growth': 0}, 'flags': [f"Error:
{str(e)}; fallback empty"], 'cached_state': {}}))

```

Output Requirements (Steps 1-3)

Proposed Addition Report: Numbered H2 list (≤8 quality-filtered); new sections by insertion; markers e.g., '[[Insert after para 2: ID-3]]'.

Artifact Summary

JSON (≤250 tokens, ```json block): {"additions": [{"Career Options": [{"pitch": "AI boosts 20%", "source_url": "ex.com", "snippet": "2025 outlook", "entity_id": "P1"}]}, ...], "new_sections": ["Trends"], "replacements_proposed": [2, e.g., "Old Bootcamp → New Bootcamp Alt"], "wc_audit": {"est_growth": "≤20%"}, "flags": ["3 filtered <0.7"], "for_next": ["Pitches/alts for P3"], "prompt4_prep": ["Markers for insertion"], "full_available": true}. >10: {'count': N, 'examples': [:3], 'full_available': true}. (Keys for chain: replacements_proposed track P3 drafts.)

3 - Generate Working Article

###Prompt 3 - Compile Working Article###

Task

Apply prior Artifacts from Prompts 1-2 (recall entities/claims/sources/additions from history). Rewrite/improve Original Article intro (1-2 paras to first H2); add Key Takeaways (5-6 bullets summarizing article for SEO/skim). Output: JSON Working Article artifact.

Preservation:

See ###Refurbishment Project Guidelines###.

Prompt 3 Tools/Notes

- **Cohesive**: Intro/KTs/body as one; minimal blends (1-5 word transitions).
- Load priors via code_execution (json.loads('recall from history')); fallback {'fallback': true}.
- **Chunk >4000 words:** chunks = [text[i:i+4000] for i in range(0, len(text), 4000)]; merge.

###Step 0: Detect Topic

Infer core theme from title/body for intro/KTs. **Run code_execution:**

```
import re, json

try:

    recall_summary = 'recall from history: P2 summary like {"detected_topic": "[topic]", "additions": {"count": 5, "examples": ["new program"]}}'  # From thread

    title = re.findall(r'"title":\s*"([^"]+)"', recall_summary)[0] if re.search(r'"title"', recall_summary) else 'Sample Title'  # Pull from recall

    body = re.findall(r'"body_snippet":\s*"([^"]+)"', recall_summary)[0][:1000] if re.search(r'"body_snippet"', recall_summary) else ''  # Pull body

    combined = (title + ' ' + body).lower()

    topic_matches = re.findall(r'bootcamp|career|financing|school|degree|veteran|dental|job|finance', combined)  # Extended for title/body

    topic = topic_matches[0] if topic_matches else 'general'
```

```
    # Fallback body-only scan if no matches
    if topic == 'general':
        body_matches = re.findall(r'job|degree|career|finance|bootcamp|school', body.lower())
        topic = body_matches[0] if body_matches else 'general'
    print(json.dumps({'detected_topic': topic}))
except Exception as e:
    print(json.dumps({'detected_topic': 'general', 'flags': [f"Error: {str(e)}; fallback general"]}))
```

###Step 1: Generate Intro

Rewrite intro (after title to first H2): 2 paras, 100-130 words total; ≤4 sentences/para. Hook with curiosity/empathy; tease outcomes (roles/salaries/paths) generally (e.g., 'high-paying roles in [topic]'). End with transition to guide.

- **Hard limit:** 1 citation-worthy claim (e.g., key stat like average cost); retro-^n uncited in body via P5 placements loop in Step 3 code.
- **Minimalism**: ≤20% growth; hooks first, facts second; trim if >130 words.

###Step 2: Generate Key Takeaways

H2 after intro: 5-6 bullets from P1 highlights + P2 pitches; generalize claims (e.g., 'high-paying' not specifics). No new data.

###Step 3: Produce the 'Working Article'

Audit intro (0-1 claim, ^n), KTs (no new info), full text (echo flags, uncited retro-^n, wc/token). Check P2 pitches applied; parse for anchors, assemble 'working article.' **Run code_execution:**

```
import re, json
try:
    recall_summary = 'recall from history: P2 summary like {"detected_topic": "[topic]", "additions": {"count": 5, "examples": ["new program"]}}'
    title = re.findall(r'"title":\s*"([^"]+)"', recall_summary)[0] if re.search(r'"title"', recall_summary) else 'Sample Title'
```

```
    body = re.findall(r'"body_snippet":\s*"([^"]+)"', recall_summary)[0] if
re.search(r'"body_snippet"', recall_summary) else input('Paste article body: ')

    wc = len(body.split())

    long_article = wc > 4000

    if long_article:

        chunks = [body[i:i+4000] for i in range(0, len(body), 4000)]

        full_text = '\n\n'.join(chunks)

    else:

        full_text = body

    combined = (title + ' ' + full_text).lower()

    topic_matches =
re.findall(r'bootcamp|career|financing|school|degree|veteran|dental|job|finance', combined)

    detected_topic = topic_matches[0] if topic_matches else 'general'

    if detected_topic == 'general':

        body_matches = re.findall(r'job|degree|career|finance|bootcamp|school', full_text.lower())

        detected_topic = body_matches[0] if body_matches else 'general'

    artifact = json.loads('recall from history') if 'recall from history' else {'working_article':
input('Paste working_article: ')}

    long_article = artifact.get('long_article', long_article)

    sections = artifact.get('sections', [{"content": "sample section text"}])

    anchors = []

    for sec in sections:

        base_regex = r'\b[A-Z][a-z]+
(Bootcamp|School|Program|Degree|Financing|Career|Scholarship)\b|\b(tuition costs|job growth
stats|salary median|growth rate|duration hours|GPA requirements|scholarship deadlines)\b'

        if detected_topic in ['career', 'degree']:

            base_regex += r'|\b(job requirements|degree duration|entry-level tips)\b'
```

```
        sec_anchors = re.findall(base_regex, sec['content'])
        anchors.extend(sec_anchors[:2])
    if len(anchors) > 12:
        anchors = anchors[:12]
    intro_draft = "Placeholder intro text from original + improvements"  # Use P1 highlights for stat, match conversational tone
    claims_in_intro = len(re.findall(r'\$[\d,]+|\d+%|ranked #\d+|GPA \d+\.\d+', intro_draft))
    if claims_in_intro > 1:
        intro_draft = re.sub(r'\$[\d,]+|\d+%', 'affordable', intro_draft)
    trimmed_intro = intro_draft[:130]
    kt_bullets = ["Placeholder KT 1 from highlights", "KT 2 from pitches"]
    kts_h2 = '\n\n## Key Takeaways\n' + '\n'.join([f'- {kt}' for kt in kt_bullets[:6]])
    intro = trimmed_intro
    body = '\n\n'.join([sec.get('snippet', '') for sec in sections])
    full_text = intro + kts_h2 + '\n\n' + body
    deletions = artifact.get('deletions', [])
    for del_entity in deletions:
        full_text = re.sub(r'\b' + re.escape(del_entity) + r'\b(?:\s*|\s*[^|]+)*', '', full_text)
    replacements = artifact.get('replacements_proposed', [])
    for rep in replacements:
        old = rep.get('old', '')
        new = rep.get('new', '')
        full_text = re.sub(re.escape(old), new, full_text)
    claims_regex = r'\$[\d,]+\.?\d*(?:/credit| /semester| /year|\s*per\s*\w+)?|\d+% (?:growth|grad|acceptance|licensure|pass|rate|discount)|\d+ (?:jobs?|openings|new jobs?|hours?|credits?|semesters?|years?|enrollments?|students?|awards?)|ranked
```

```
#\d+|#\d+|GPA \d+\.\d+|\d+-\d+ (?:credits?|hours?|jobs?|tuition|salary|costs?)|average (?:\$[\d,]+\.?\d*|\d+%|\d+ (?:jobs?|hours?|credits?))|costs range from \$[\d,]+ to \$[\d,]+|most affordable (?:\$[\d,]+|\d+ credits?)|(?:as low as|projected|estimated) (?:\$[\d,]+|\d+%|\d+ (?:credits?|hours?))'

    placements = artifact.get('claim_placements', [])

    uncited = re.findall(claims_regex, full_text)

    for unc in uncited:

        pos = full_text.find(unc)

        if pos != -1 and '^' not in full_text[pos:pos+50]:

            matching_n = next((p['n'] for p in placements if unc in p['claim']), None)

            if matching_n:

                full_text = re.sub(re.escape(unc) + r'(?=\s*[.!?,;])', f'{unc}^{matching_n}', full_text, count=1)

    p2_inserts = len(re.findall(r'\[\[Inserted:', full_text))

    p2_pitches = len(artifact.get('additions', {'count': 0})['count'])

    p2_flag = ['P2 pitches missed'] if p2_inserts < p2_pitches else []

    if long_article:

        chunks = re.split(r'(?=<h2>|^## )', full_text)

        working_article = [chunk.strip() for chunk in chunks if chunk.strip()]

        if len(working_article) > 4:

            working_article = working_article[:4]

    else:

        working_article = full_text

    print(json.dumps({'detected_topic': detected_topic, 'intro_wc': len(intro.split()), 'kt_bullets': len(kt_bullets), 'anchors': list(set(anchors)), 'working_article': working_article, 'p2_flag': p2_flag, 'cached_state': {'anchors': list(set(anchors)), 'working_article': working_article}}))

except Exception as e:
```

```
    print(json.dumps({'detected_topic': 'general', 'intro_wc': 0, 'kt_bullets': 0, 'anchors': [], 'working_article': 'fallback string', 'p2_flag': [], 'cached_state': {}, 'flags': [f"Error: {str(e)}"]}))

```

###Step 4: Editor Approval

- For zero-input runs, (no flags in Step 4) auto-approve. "If 'batch=true' in article text, auto-approve ≤3 flags.
- If (and only if) Step 4 audit flags any issues (e.g., growth/shrink deviations, uncited claims), **do not proceed**. Instead, generate the **Flagged Sections Report**:
    - Markdown Table: | H2/H3 | Flagged Issue | Recommended Fix |.

Output Requirements (Steps 1-4)

Artifact Summary

Embed all (intro, KTs, body with inserts/chunks) in 'working_article' string; no separate Markdown/narrative outside JSON." explicitly directs the AI to nest everything (rewritten intro, KTs H2/bullets, body with [[Inserted:]] markers, chunks if long) as a single string value in the JSON's 'working_article' key. No outside sections—output is pure JSON (block, ≤500 tokens) like: {"detected_topic": [e.g., "career"], "intro_wc": 120, "kt_bullets": 6, "citation_count": 1, "long_article": true/false, "chunks": ["Part 1: Intro/KTs", "Part 2: Mid", "Part 3: Lists", "Part 4: End"] if long, "for_next": ["Insert intro/KTs into Working Article"], "prompt3_insert": ["Place intro before first H2; KT as new H2 after intro"]}, "sections": [...], "changes": [...], "echo_dict": [...], "wc_audit": {...}, "flags": [...], "for_next": [...], "prompt5_prep": [...], "assembly_map": {...}, "full_available": true, "p2_inserts": [count]}.

- Compress snippets to ≤20 words; if sections >10, summarize {'count': len, 'examples': [:3]}.
- working_article Format: For long_article=true, 'working_article': array of chunk strings (e.g., ["Part 1: Intro + KTs\n\n[content]", "Part 2: ..."]); for short, single string. Wrap output in ``` for clarity.

4 - SEO Improvements

###Prompt 4 — SEO & Links###

Task

Load P3 Artifact (code_execution json.loads('recall from history'); use sections/anchors/changes/sources). Propose SEO first-sentence rewrites + JB internal links. No fact changes/external sources. **Output**: SEO Report + Artifact Summary.

Preservation

See ###Refurbishment Project Guidelines###

Prompt 4 Tools/Notes

- **Links Verify:** web_search_with_snippets 'site:jobbuilder.com "[anchor]" exact' (num=5; HTTP 200/snippet). Fallback browse_page inferred URL ('Verify 200, title/content [theme]').
- **Self-Critique:** JB-only; ≤2 chains (browse + broaden 'site:jobbuilder.com [anchor] guide OR how-to -[topic]' for diversity, e.g., /blog/fafsa). Propose ≥1 live; flag none/unverified. Adapt [topic] to core (e.g., 'bootcamps' vs. 'degrees'); >80% unique slugs (flag <80%).
- **Fallback:** json.loads fail → {'fallback': true}; manual parse. If topic missing: code topic = re.findall(r'bootcamp|career|...|degree', title.lower())[0] or 'general'.

Task Sequence

###Step 1: Propose SEO Upgrades

First-Sentence Rewrites: ≤20 words per H2/H3; SEO-friendly, header-answering. Conversational/casual voice; no marketing ('Discover...'). Preserve claims. Maximum one re-write per H2/H3.

- Preserve quirks/repetitions.
- First body sentence only; ≤20 words, header unchanged.
- Diversify syntax (not all noun-starts); optionally include 0-1 SEO keyword insertions for anchor potential in Step 2 (e.g., 'affordable tuition' if fits verbatim, no fact changes or larger re-writes); natural fits only.

###Step 2: Internal Links

Propose JB articles for SEO boost. Verify any original links in the article using browse_page; if broken, propose updates from JB site only, without adding new externals.

- **Parse Working Article anchors** (verbatim entities/SEO terms, e.g., "tuition costs");
  Compile web_search broadened for URLs (e.g., /schools/[name]); vary theme (entities → /schools, general → /blog/how-to);
  De-dupe unique slugs; prioritize schools; no dupes (>20% same → flag).
    - If duplicates, replace or skip link. For each link anchor, if matches P5 claim, append ^n (e.g., anchor^n); prioritize primary-sourced slugs from P1 sources.

Run code_execution (anchors/keywords/variety/audit):

```
import re, json

try:

    recall_summary = 'recall from history: P3 summary like {"detected_topic": "[topic]", "sections": [{"title": "[section title]", "snippet": "[first 50 words]"}], "working_article_snippet": "[intro text]"}'

    try:

        artifact = json.loads('{"anchors": [], "detected_topic": "general"}')

        artifact5 = json.loads('{"sources": [], "claim_placements": []}')

    except json.JSONDecodeError:

        artifact = {"anchors": [], "detected_topic": "general"}

        artifact5 = {"sources": [], "claim_placements": []}

    working_snippet = re.findall(r'"working_article_snippet":\s*"([^"]+)"', recall_summary)[0][:1000] if re.search(r'"working_article_snippet"', recall_summary) else input('Paste working_article: ')

    sections_snippet = re.findall(r'"sections":\s*\[\s*{"title":\s*"([^"]+)",\s*"snippet":\s*"([^"]+)"', recall_summary)[:3]

    working_text = working_snippet + ' '.join([s[1] for s in sections_snippet])

    detected_topic = artifact.get('detected_topic', 'general')

    if detected_topic in ['career', 'entry-level', 'veteran']:

        anchors_regex = r'\b[A-Z][a-z]+ (Bootcamp|School|Program|Degree|Financing|Veteran|Hygiene|Job)\b|\b(tuition costs|salary median|scholarship basics|fafsa guide)\b'

    else:
```

```
    anchors_regex = r'\b[A-Z][a-z]+ (Bootcamp|School|Program|Degree|Financing)\b|\b(tuition costs|fafsa guide)\b'

    anchors = re.findall(anchors_regex, working_text)

    verified_anchors = [a for a in artifact.get('anchors', []) if a in working_text]

    original_links = re.findall(r'\[([^\]]+)\]\s*\((https?://[^\)\s]+)\)', working_text)

    verified_links = []

    for text, url in original_links:

        if 'jobbuilder.com' in url and True:  # Simulate: browse_page(url)

            verified_links.append((text, url))

        else:

            # Simulate: web_search_with_snippets(f'site:jobbuilder.com "{text}" exact', num=5)

            verified_links.append((text, f'https://jobbuilder.com/blog/{text.lower().replace(" ", "-")}'))

    seeds = verified_anchors

    text_snippet = working_text

    if len(text_snippet) > 5000:

        chunks = [text_snippet[i:i+2000] for i in range(0, len(text_snippet), 2000)]

        expanded = sum((re.findall(r'\b(tuition costs|fafsa guide)\b' + ('|scholarship|loan' if 'financing' in detected_topic else ''), chunk) for chunk in chunks), []) if len(verified_anchors) < 12 else []

        verified = [a for a in seeds if any(a in chunk for chunk in chunks)]

    else:

        expanded = re.findall(r'\b(tuition costs|fafsa guide)\b' + ('|scholarship|loan' if 'financing' in detected_topic else ''), text_snippet) if len(verified_anchors) < 12 else []

        verified = [a for a in seeds if a in text_snippet]

    verified += expanded[:12 - len(verified)]

    final_anchors = list(set(verified))
```

```
    keywords = re.findall(r'\b(affordability|tuition|career|fafsa|scholarship basics)\b', working_text)
    # Simulate: web_search_with_snippets(f'site:jobbuilder.com "{detected_topic}"', num=5)
    unique_urls = list(set([url for _, url in verified_links] +
[f'https://jobbuilder.com/blog/{detected_topic.lower()}']))
    seen = set()
    filtered_urls = []
    for url in unique_urls:
        slug = url.split('/')[-1]
        if slug not in seen:
            seen.add(slug)
            filtered_urls.append(url)
    unique_urls = filtered_urls
    p5_placements = artifact5.get('claim_placements', [])
    sourced_urls = []
    for url in unique_urls:
        anchor = url.split('/')[-1]
        matching_n = next((p['n'] for p in p5_placements if anchor in p['claim'].lower()), None)
        if matching_n:
            sourced_urls.append(f"[{anchor}]({url})^{matching_n}")
        else:
            sourced_urls.append(f"[{anchor}]({url})")
    unique_urls = sourced_urls
    variety_score = len(set(unique_urls)) / len(unique_urls) if unique_urls else 1
    flags = ['Low variety (<80% unique slugs)'] if variety_score < 0.8 else []
    cached = {'verified_anchors': final_anchors, 'seo_keywords': keywords}
```

```
    print(json.dumps({'verified_anchors': final_anchors, 'count': len(final_anchors), 'seo_keywords': keywords, 'unique_urls': unique_urls, 'variety_score': variety_score, 'flags': flags, 'cached_state': cached}))

except Exception as e:

    print(json.dumps({'verified_anchors': [], 'count': 0, 'seo_keywords': [], 'cached_state': {}, 'flags': [f"Error: {str(e)}; fallback empty"]}))
```

Output Requirements (Steps 1-2)

SEO Report: Per H2/H3: Rewrite ("Already Optimized" or ≤20 words); anchor + JB link. No link: 'No verified (reason: dupes)'. Ex: 'Link "roles" para 1 to /blog/jobs (site:verified)'. Flag >20% same slug. Markdown list; no narrative. Format SEO Report with headers: ## SEO Sentence Rewrites and ## Internal Link Proposals for clarity.

Artifact Summary

JSON (≤250 tokens, ```json block): {"additions": [{"Career Options": [{"pitch": "AI boosts 20%", "source_url": "ex.com", "snippet": "2025 outlook", "entity_id": "P1"}]}, ...], "new_sections": ["Trends"], "wc_audit": {"est_growth": "≤20%"}, "flags": ["3 filtered <0.7"], "for_next": ["Pitches for P3"], "prompt4_prep": ["Markers for insertion"], "full_available": true}. >10: {'count': N, 'examples': [:3], 'full_available': true}. (Keys for chain: additions/sources propagate.)

5 - Final Sources List

###Prompt 5 — Citation Clean-Up###

Task

Reference P1-4 Artifacts (recall Working Article/additions/anchors/sources). Consolidate sources, renumber ^n sequentially by appearance, merge echoes. Preview cleaned Sources. **Output**: Sources Section + Artifact Summary.

Recall from History

Summarize prior artifacts as mini-JSON (e.g., prior: {'detected_topic': '[topic]', 'sections': [{'title': '[section title]', 'wc': 85, 'snippet': '[first 50 words]'}], 'working_article_snippet': '[intro text]'}). Use for loads/parsing.

Preservation

See ###Refurbishment Project Guidelines###.

Prompt 5 Tools/Notes

No new edits/rewrites/additions except for citation placeholders (^n) and generated Sources H2.

- No new content; strip params/broken; failed sources → web_search_with_snippets '[query] [Date]'; none = evergreen.
- **Echoes:** Guidelines def (exact quant match); reuse ^n.
- **Re-Verify All Claims:** For each P1 claim, run web_search_with_snippets 'claim text update Oct 2025 site:bls.gov OR ada.org OR official.edu' (num=3); if snippet ≥93% match (self-critique: exact $/% + date), update source/snippet; else fallback 'Recent data [contact school]' + evergreen flag. Parallel ≤3 calls; omit if no primary.
- **Load:** P1-3 JSONs for claims/sources; scan full P3 working_article (reassemble chunks if long).
- **Fallback:** json.loads fail → {'fallback': true}; manual parse.

Task Sequence

###Step 1: Citations Cleanup

Cleanup internally (no full output). Include P4 links if claim-supporting. Tables: Shared ^n/row note. **Code Logic**: De-dup sources (P1/P2/P3 raw); scan full text top-down regex (intro>body>tables); merge echoes to first ^n (body>table; primary>secondary); retro-^n uncited if match; placement (.^n after punct, * for echoed cells); sequential 1-N no gaps. Flag

orphans/deviations. For claims in tables, apply shared ^n per row or column as note, using * for echoed cells to avoid repetition.

Run code_execution (full cleanup; chunk >40 sources):

```

import json, re

try:

    recall_summary = 'recall from history: P1-4 summaries like P1 {"claims": {"count": 20}}, P3 {"working_article": "[full or chunks]"}, P4 {"unique_links": 5}}'

    try:

        artifact1 = json.loads('{"claims": [], "sources": [], "echo_dict": {}, "claim_placements": []}')

        artifact2 = json.loads('{"additions": []}')

        artifact3 = json.loads('{"intro_wc": 0, "citation_count": 0, "claims": [], "working_article": "full text or chunks"}')

        long_article = artifact3.get('long_article', False)

        if long_article and isinstance(artifact3['working_article'], list):

            working_article = '\n\n'.join(artifact3['working_article'])

        else:

            working_article = artifact3.get('working_article', input('Paste working_article: '))

        snippet = working_article[:3000] if not long_article else working_article

    except json.JSONDecodeError:

        artifact1 = {"claims": [], "sources": [], "echo_dict": {}, "claim_placements": []}

        artifact2 = {"additions": []}

        artifact3 = {"intro_wc": 0, "citation_count": 0, "claims": [], "working_article": input('Paste working_article: ')}

        long_article = False

        working_article = artifact3['working_article']
```

```
        snippet = working_article

    if 'full_available' in artifact1 and artifact1['full_available']:

        report_text = 'prior P1 report Markdown here'

        full_claims = re.findall(r'\| ID \| Location \| Original Claim \| Updated Claim \| Source URL \| Snippet \| Echo: (.+?) \|?', report_text)

        artifact1['claims'] = full_claims

    cached = {}

    sources_pool = list(set([s.get('source_url', '') for s in (artifact1.get('claims', []) + artifact3.get('claims', []))] + [a.get('source_url', '') for a in artifact2.get('additions', []) if 'source_url' in a]))

    claims = [c.get('claim', '') for c in artifact1.get('claims', []) if c.get('status') == 'verified'] + [c.get('claim', '') for c in artifact3.get('claims', [])]

    echo_dict = artifact1.get('echo_dict', {})

    placements = artifact1.get('claim_placements', [])

    renumbered = {}

    current_n = 1

    claims_regex = r'\$[\d,]+\.?\d*(?:/credit| /semester| /year|\s*per\s*\w+)?|\d+% (?:growth|grad|acceptance|licensure|pass|rate|discount)|\d+ (?:jobs?|openings|new jobs?|hours?|credits?|semesters?|years?|enrollments?|students?|awards?)|ranked #\d+|#\d+|GPA \d+\.\d+|\d+-\d+ (?:credits?|hours?|jobs?|tuition|salary|costs?)|average (?:\$[\d,]+\.?\d*|\d+%|\d+ (?:jobs?|hours?|credits?))|costs range from \$[\d,]+ to \$[\d,]+|most affordable (?:\$[\d,]+|\d+ credits?)|(?:as low as|projected|estimated) (?:\$[\d,]+|\d+%|\d+ (?:credits?|hours?))'

    all_matches = []

    table_regex = r'\|.*?[\n\|]+.*?(?:\n|$)'

    tables = re.findall(table_regex, working_article, re.DOTALL)

    for m in re.finditer(claims_regex, working_article):

        claim = m.group(0)

        if claim in claims:
```

```
            url = next((s.get('source_url') for s in (artifact1.get('claims', []) + artifact3.get('claims', [])) if s.get('claim') == claim), None)

            if url:

                is_table = any(m.start() >= t.start() and m.end() <= t.end() for t in re.finditer(table_regex, working_article))

                all_matches.append((m.start(), claim, url, is_table))

    all_matches.sort(key=lambda x: x[0])

    verified_claims = {}

    for i, (pos, claim, old_url, is_table) in enumerate(all_matches):

        if i % 3 == 0:

            match_score = 0.95  # Simulated

            if match_score >= 0.93:

                verified_claims[claim] = {'url': 'https://example.com', 'snippet': '2025 data verified'}

            else:

                verified_claims[claim] = {'url': 'Contact school for 2025 rates', 'snippet': 'Recent data; evergreen flag', 'fallback': True}

    for pos, claim, old_url, is_table in all_matches:

        new_data = verified_claims.get(claim, {'url': old_url, 'snippet': ''})

        url = new_data['url']

        snippet_text = new_data['snippet']

        if url not in renumbered:

            renumbered[url] = {'n': current_n, 'url': url, 'snippet': snippet_text, 'echoes': [pos], 'claim': claim, 'fallback': new_data.get('fallback', False), 'is_table': is_table}

            current_n += 1

        else:

            renumbered[url]['echoes'].append(pos)
```

```
            if is_table:
                working_article = re.sub(re.escape(claim) + r'(?=\s*[.!?,;]|\|)', f'{claim}*',
working_article, count=1)
            else:
                working_article = re.sub(re.escape(claim) + r'(?=\s*[.!?,;])',
f'{claim}^{renumbered[url]["n"]}', working_article, count=1)
    for place in placements:
        claim = place['claim']
        expected_n = place['n']
        locations = place['locations']
        actual_count = len(re.findall(re.escape(claim), working_article))
        if actual_count < len(locations):
            flags.append(f"Missed echo for {claim}: Expected {len(locations)}, found {actual_count}")
        if actual_count > 0 and expected_n not in [v['n'] for v in renumbered.values() if claim in
v['claim']]:
            working_article = re.sub(re.escape(claim) + r'(?=\.|$)', f'{claim}^{expected_n}',
working_article)
            flags.append(f"Retro-validated ^ {expected_n} for {claim}")
        for loc in locations:
            if claim not in working_article:
                flags.append(f"Orphan placement: {claim} at {loc}")
    echo_merges = sum(len(v['echoes']) - 1 for v in renumbered.values() if len(v['echoes']) > 1)
    flags = ['Orphan claim: ' + c for c in set(claims) if c not in working_article] + [f"Retro-cited
{len(all_matches) - len(renumbered)} echoes"]
    cached['sources'] = [{'n': v['n'], 'url': k, 'snippet': v['snippet'], 'echoes': len(v['echoes']), 'claim':
v['claim']} for k, v in renumbered.items()]
    cached['echo_merges'] = echo_merges
```

```
    claim_placements = [{'claim': v['claim'], 'locations': len(v['echoes']), 'n': v['n']} for v in renumbered.values()]

    cached['claim_placements'] = claim_placements

    print(json.dumps({'sources': cached['sources'], 'renumbered_n': current_n - 1, 'echo_merges': echo_merges, 'flags': flags, 'claim_placements': claim_placements, 'working_article_updated': working_article[:1000] + '...' if len(working_article) > 1000 else working_article, 'cached_state': cached}))

except Exception as e:

    print(json.dumps({'sources': [], 'renumbered_n': 0, 'echo_merges': 0, 'flags': [f"Error: {str(e)}; fallback empty"], 'claim_placements': [], 'cached_state': {}}))
```

###Step 2: Editor Approval (if Necessary)

Auto-fix minors (typos, <5 dangling/uncited) via Step 1 code. If major flags (growth dev, P2 miss, >5 orphans): **Output Flagged Report** (Markdown | Issue | Fix |); await 'approve all'/'reject IDs' for Sources.

Output Requirements (Steps 1-2)

- **Sources List:** Markdown numbered '1. [URL]. Echo #s: [^1,^5]. Snippet: [≤20 words]'. Sequential by appearance (^1 first); shared ^n same URL; no full article.
    - 1. https://URL (for [claim]) to include claim context.
- **Chunk >40:** 'Sources 1-40' ‘Sources 41-80,’ etc.; reassemble on approval.

Artifact Summary

JSON (≤250 tokens, ```json): {"sources": [{"n": 1, "url": "ex.com", "echoes": [1,5], "snippet": "2025 BLS", "status": "verified", "first_appearance": "intro para 1"}, ...], "renumbered_n": [15], "echo_merges": ["5 consolidated"], "flags": ["3 retro-cited"], "for_next": ["^n/placements for P6"], "full_available": true, "updated_working_article": "snippet/flag load"}. >10: {'count': N, 'examples': [:3], 'full_available': true}. (Keys for chain: sources/echoes propagate.)

6 - Final Assembly in HTML

###Prompt 6 — Final Assembly###

Task

Reference P1-5 Artifacts (recall Working Article/SEO/Sources). Apply P5 ^n/sources, P4 rewrites/links to Working Article; format as Markdown (Docs paste compatibility). Remove markers ([[Inserted]], strikethrough); seamless flow. **Output**: Full Markdown code block.

Preservation

See ###Refurbishment Project Guidelines###.

Prompt 6 Tools/Notes

Integrate and format only — no new content.

- No new content; standardize **bold**/*italics*; preserve originals (salaries/tables).
- **Markdown:** # H1, ## H2, ### H3, - bullets, | tables, [anchor](full URL) links, ^n citations.
- **Load Fallback:** json.loads fail → {'fallback': true}; manual parse.

Task Sequence

###Step 1: SEO and Formatting

Insert P4 rewrites (first sentence/H2/H3; header unchanged); embed links as [anchor](full URL). Replace P5 ^1-N with ^n. Flag mismatches '[AUDIT: ^n not applied]'. Auto-resolve minors via code (regex dangling ^n to nearest source; flag >3). For chunked outputs, label as Part X/N: [H2 title] using first H2 in chunk (extract via re.findall(r'^##\s+([^\n]+)', chunk, re.MULTILINE)).

- Scan uncited (broadened regex for lists: \$[\d,]+\.?\d*(?:/credit| /semester| /year|\s*per\s*\w+)?|\d+ (?:credits?|semesters?|years?)|\d+% (?:growth|grad|acceptance)) → retro-^n from P5 via loop: for each match, sub claim + '^{n}' from P5 if source matches.
- Embed links inline in sentences where anchors appear (e.g., 'structure. See dental schools for more.'); no separate lines.
- Retro-^n: P5 placements (re.sub claim + '^{n}').
- Alignment: Dangling → '[EDITOR: ^n no source]'; uncited → '[EDITOR: Add ^n]'.
- No strikethroughs; minimal newlines.
- Cleaning: Run code_execution pre-assembly:

```
import re
# Assume working_article and placements are defined from prior (e.g., from artifact load)
```

```
if 'working_article' not in locals():
    working_article = input('Paste working_article: ')
if 'placements' not in locals():
    placements = json.loads(input('Paste P5 placements JSON: ')) if input('P5 placements
available? (y/n): ') == 'y' else []
if isinstance(working_article, list):
    working_article = '\n\n'.join(working_article)
    print(f'Reassembled length: {len(working_article)} chars')
working_article = re.sub(r'\[\[.*?\]\]|~~.*?~~|<!--.*?-->', '', working_article, flags=re.DOTALL)
working_article = re.sub(r'\n{3,}', '\n\n', working_article)
working_article = re.sub(r'\n{2,}', '\n', working_article)
working_article = re.sub(r'(\[.*?\]\(https?://[^\)\s]+\))', r' \1', working_article)
uncited = re.findall(r'\$[\d,]+\.?\d*(?:/credit| /semester| /year|\s*per\s*\w+)?|\d+%
(?:growth|grad|acceptance|licensure|pass|rate|discount)|\d+ (?:jobs?|openings|new
jobs?|hours?|credits?|semesters?|years?|enrollments?|students?|awards?)|ranked
#\d+|#\d+|GPA \d+\.\d+|\d+-\d+ (?:credits?|hours?|jobs?|tuition|salary|costs?)|average
(?:\$[\d,]+\.?\d*|\d+%|\d+ (?:jobs?|hours?|credits?))|costs range from \$[\d,]+ to \$[\d,]+|most
affordable (?:\$[\d,]+|\d+ credits?)|(?:as low as|projected|estimated) (?:\$[\d,]+|\d+%|\d+
(?:credits?|hours?))', working_article)
for u in uncited:
    matching_n = next((p['n'] for p in placements if u in p['claim']), None)
    if matching_n:
        working_article = re.sub(re.escape(u) + r'(?=\s*[.!?,;])', f'{u}^{matching_n}', working_article,
count=1)
if len(working_article.split()) > 5000:
    chunks = re.split(r'(?=<h2>|^## )', working_article)
    labeled_chunks = []
    for i, chunk in enumerate(chunks[:4], 1):
        h2 = re.findall(r'^##\s+([^\n]+)', chunk, re.MULTILINE)
        label = f'Part {i}/{min(len(chunks), 4)}: {h2[0] if h2 else "Section"}'
        labeled_chunks.append(f'{label}\n\n{chunk.strip()}')
    print('\n\n'.join(labeled_chunks))
else:
    print(working_article[:1000] + '...' if len(working_article) > 1000 else working_article)
```

###Step 2: Sources

End with P5 list as Markdown numbered: 1. https://URL (plain URL, auto-links in Docs; no [ ] wrapper for clean display).

###Step 3: Internal Audit

Audit text (wc, structure, cites, growth). **Run code_execution** (Markdown-tuned):

```
import re, json

try:

    recall_summary = 'recall from history: P3-5 summaries like P3 {"sections": [{"title": "[section title]", "snippet": "[first 50 words]"}], "wc_audit": {"growth": 12.5}}, P5 {"sources": {"count": 15}, "claim_placements": []}'  # From thread

    cached = {}

    # Pull md from recall

    md = re.findall(r'"working_article":\s*"([^"]+)"', recall_summary)[0] if re.search(r'"working_article"', recall_summary) else 'full Markdown text from history'

    if isinstance(md, list):

        md = '\n\n'.join(md)

    try:

        artifact3 = {"sections": [], "wc_audit": {"og_wc": 0}}  # Fallback dict

        artifact4 = {"unique_links": []}

        artifact5 = {"sources": [], "echo_merges": 0, "claim_placements": []}

        # Auto-retry: if 'recall

    except:

        artifact3 = {"sections": [], "wc_audit": {}}

        artifact4 = {"unique_links": []}

        artifact5 = {"sources": [], "echo_merges": 0, "claim_placements": []}

    # 1: Clean/Count (Markdown)

    md = re.sub(r'\s+', ' ', md)

    md = re.sub(r'\\|"escaped quotes"', '', md)

    wc = len(re.sub(r'#|*|||.*? $$$$ .*?|^\d+. |^\d+', '', md).split())  # Strip MD + ^n

    token_est = wc * 4
```

```
    original_wc = artifact3['wc_audit'].get('og_wc', wc)

    growth = ((wc - original_wc) / original_wc * 100) if original_wc else 0

    # 2: Structure

    additions_present = bool(re.search(r'P2 addition|^\- ', md))

    kt_placement = bool(re.search(r'## Key Takeaways\n\- ', md))

    deletions_applied = len(re.findall(r'strikethrough|\~\~.*?\~\~', md)) == 0

    structural_flags = [] if (additions_present and kt_placement and deletions_applied) else
['AUDIT: P2 missing', 'AUDIT: KT off', 'AUDIT: Deletions not applied']

    # 3: Citations (^n)

    citations = [int(c[1:]) for c in re.findall(r'\^(\d+)', md)]

    sources = [s['n'] for s in artifact5['sources']]

    dangling = [c for c in citations if c not in sources]

    uncited_claims = [u for u in re.findall(r'\$[\d,]+|\d+%|ranked #\d+|GPA \d+\.\d+', md) if not
re.search(r'\^\d+', md[md.find(u):md.find(u)+50])]

    placements_applied = len(artifact5.get('claim_placements', [])) == len([p for p in
artifact5['claim_placements'] if f'^{p["n"]}' in md])

    cite_flags = [f'Dangling ^n: {d}' for d in dangling[:3]] + ['Uncited stat: ' + u for u in
uncited_claims[:3]] + (['Placements not applied'] if not placements_applied else [])

    # 4: Links/Growth ([text](url))

    applied_links = len(re.findall(r'$$ .*? $$$$ .*? $$', md))

    growth_flag = ['Growth >20%'] if growth >20 else ['Shrink >7%'] if growth < -7 else []

    # 5: Sources match

    max_n = max(citations, default=0)

    sources_count = len(artifact5['sources'])

    mismatch_flag = ['Sources/ ^n mismatch'] if max_n != sources_count else []

    flags = structural_flags + cite_flags + growth_flag + mismatch_flag + [f'Applied links:
{applied_links}']
```

```
    # Cache

    cached['final_wc'] = wc

    cached['flags'] = flags

    cached['token_est'] = token_est

    print(json.dumps({'final_wc': wc, 'applied_links': applied_links, 'flags': flags, 'token_est': token_est, 'cached_state': cached}))

except Exception as e:

    print(json.dumps({'final_wc': 0, 'applied_links': 0, 'flags': [f"Error: {str(e)}; fallback empty"], 'token_est': 0, 'cached_state': {}}))
```

###Step 4: Editor Approval

If Step 3 flags: Flagged Report | H2/H3 | Issue | Fix |; await approval. No flags: '[AUDIT: Clean]'; proceed to Markdown.

Output Requirements (Steps 1-3)

Full Markdown: # H1, ## H2, ### H3, - bullets, | Col | --- tables, [anchor](full URL) links, ^n citations. Wrap ``` block. No XML/escapes; en dash – (no —).

- Links: [anchor](full url)
- Sources: End numbered 1. https://URL (plain, auto-links).
- Split >80% limit (code est.): 3-5 chunks (~5k/part, logical); resume start_pos=len(previous); label 'Part X/4: Title'. Final: 'Reassembled: Paste 1-4 for Complete Markdown'.
- No meta; snippets ≤20 words.

Token Limit Handling: Before output, use code_execution to count tokens (e.g., Python len(text.split()) * 1.3 for Markdown overhead); log 'Split initiated at {token_est} tokens'; if >70% of estimated limit (e.g., 20k tokens total), auto-split into 3-5 fixed ~5,000-7,500-token chunks by logical sections (e.g., Part 1: Intro/Key Takeaways to end of first major section; Part 2: Second major section; Part 3: Third major section; Part 4: Final sections/FAQ/Sources). Enforce min 5k tokens/part; cut at sentence-end/post-table if mid-section.

    - **Reassembly:** On final part, add 'Full Article Reassembled: Paste Parts 1-4 in Order for Complete Markdown'. For resume: previous_parts = 'prior parts concatenated'; start_pos = len(previous_parts); chunk = full_md[start_pos:start_pos+7500]; label 'Part X/4: [Title]'.