# AI Drafting Prompt Library

**What this is.** The synthesis layer's working prompts: candidate drafting for each article type, dedupe/conflict checking, update-DIFF proposals, gap-cluster summarization, and metadata extraction for legacy content — plus the evaluation protocol that tells you whether to trust any of them.

**When you need it.** When you're ready to stop authoring and start curating (the Level 2→3 climb) — after Module 2's templates exist and before Module 4's gate has anything to judge.

**Which maturity level it serves.** Installs Level 3; the DIFF and gap prompts serve Level 4.

---

## 1. This is the living module

Everything else in the toolkit is built to still be true five model generations from now. This module is deliberately not: prompts are the one component that ages with the model landscape, so they are quarantined here, version-stamped ( `prompts-v1` ), and updated free for buyers as models shift. That's not a disclaimer — it's the boring-core/smart-edge architecture applied to the product itself. Your system of record, gates, and economics never change when you swap the drafting model; you re-run the evaluation protocol and, at most, take a prompt update.

**Model-agnostic by design.** These prompts assume only a competent instruction-following model with a decent context window — cloud API, local model, or whatever your platform embeds. Nothing in them names a vendor. Where behavior differs across models, the tuning notes say what to watch, not which brand to buy.

## 2. The rules baked into every prompt

Read these once so you recognize them; they're why the outputs are gateable:

1. **Grounding is absolute.** The draft may contain only what the source material supports. Anything the source doesn't establish is written as `[NEEDS-VERIFICATION: …]` — never invented. An honest hole beats a plausible guess; holes are cheap at the gate, hallucinations are expensive after it.
2. **Refusal over improvisation.** If the source can't support a draft (no resolution present, contradictory outcomes), the prompt instructs the model to output `INSUFFICIENT SOURCE` plus what's missing — a five-second reject for the curator instead of a five-minute unpicking.
3. **The customer's language survives.** Symptom phrasings and exact error text are quoted, not paraphrased into support-speak.
4. **PII is scrubbed at synthesis.** Names, emails, account ids, order numbers, tokens — replaced with neutral placeholders before the draft ever reaches the queue. The gate still checks (R4), but the prompt is the first line.

5. **Output is the contract.** Every prompt emits a complete candidate file in Module 2′s exact format — frontmatter and sections — so candidates are machine-checkable and the curator reviews content, not formatting.
6. **Sections are self-contained.** The machine-legibility rules from Module 2 §4 are restated inside the prompts, because drafts inherit their habits from their instructions.

## 3. The prompts

| File | Job | Feeds |
|---|---|---|
| `prompts/synthesis-issue-resolution.md` | Resolved case → issue/resolution candidate | Module 4's new-candidate gate |
| `prompts/synthesis-how-to.md` | Guidance interactions or release material → how-to candidate | New-candidate gate |
| `prompts/synthesis-faq.md` | Question cluster → FAQ candidate | New-candidate gate |
| `prompts/synthesis-known-error.md` | Confirmed-defect case set → known-error candidate | New-candidate gate |
| `prompts/dedupe-check.md` | Candidate vs. existing corpus → duplicate/conflict verdict | Pre-gate screen (cuts R2 load) |
| `prompts/update-diff-proposal.md` | Changed fact + affected article → surgical before/after diff | Module 4's update-DIFF gate |
| `prompts/gap-cluster-summary.md` | Raw unanswered queries → named gap clusters with counts | Module 6's Gap Log, then capture |
| `prompts/metadata-extraction.md` | Legacy untyped article → typed frontmatter proposal | Corpus migration to Module 2's schema |

Each file states what to feed it, what comes back, and its tuning notes. Slots are `{{DOUBLE_BRACED}}`; fill every slot — an empty slot is the most common cause of a bad draft.

**The `{{ORG_CONTEXT}}` contract.** This is the one slot with a required shape, because the drafts fill four frontmatter fields from it: your product names and version scheme, your **domain taxonomy** (the allowed `domain` values from Module 2′s schema, one line each with a one-phrase definition), and your allowed tag list. A context block without a domain taxonomy forces the model to guess a required field — and Module 4′s gate checks `domain` on arrival, so guesses surface as rejects.

**Source preparation** (the part people skip): a synthesis prompt is only as good as its payload. Ticket thread → include the full agent-customer exchange plus any internal resolution note, newest last. Chat transcript → include timestamps trimmed, speaker labels kept. Call transcript → run your transcription first; feed the transcript plus the wrap-up note. Release notes → feed the note plus the feature′s doc stub if one exists. Export mechanics per stack: Zendesk (ticket export or API `comments`), Salesforce (Case + CaseComments +

EmailMessages), Freshdesk (conversations export). Strip signatures and legal footers; they poison customer-language extraction.

## 4. The evaluation protocol — run this before you trust anything

Do not wire a synthesis prompt into your flow on vibes. One hour of evaluation buys you a calibrated yes/no:

1. **Freeze an eval set.** Pick 10 historical resolved cases from the pilot domain — real, closed, representative (mix of clean and messy threads). These 10 never change; they're your regression suite.
2. **Run the prompt** on all 10 with your chosen model. Record model name/version and `prompts-v` stamp on every output.
3. **Score each draft 1–5 on five axes**: factual fidelity to the source · template compliance (would `validate.py` pass it?) · customer-language preservation · applicability extraction (product/version right?) · PII cleanliness (this axis is pass/fail — one leak = 1).
4. **The bar**: median ≥ 4 on every axis, zero PII failures, and at least 7 of 10 drafts a curator would A1/A2 rather than R3. Below the bar → fix the *payload preparation* first (it's the usual culprit), then tune the prompt, then consider a different model. In that order.
5. **Re-run on every change** — new model, new prompt version, new domain. Compare against the frozen set's previous scores. This is the same discipline as the edit-distance trend (Module 4 §3), applied before production instead of during it.

Keep a one-line log per run: date, model, prompts-v, five medians, verdict. Three columns of history will one day save you from a silent model-quality regression — the failure mode nobody notices until customers do.

## 5. Tuning, generally

- **Drafts too long** → the source thread rambles; trim the payload to the resolution arc, don't fight it in the prompt.
- **Support-speak creeping in** → your examples matter more than your instructions; add one gold-standard example (Module 2's `examples/`) to the prompt as a style anchor.
- **Wrong type chosen** → don't ask one prompt to pick the type; you route (by trigger source), the prompt drafts. Type-per-prompt is deliberate.
- **Applicability guessed** → tighten `{{ORG_CONTEXT}}`: list your real product names, version scheme, and domain taxonomy so the model extracts instead of inventing.

**What this module assumes from its siblings:** Module 2's templates (the output contract) and examples (style anchors); Module 4's gate downstream of every output. Reject-reason telemetry (R2/R3 spikes) is this module's feedback signal — see the code table in Module 4 §2.