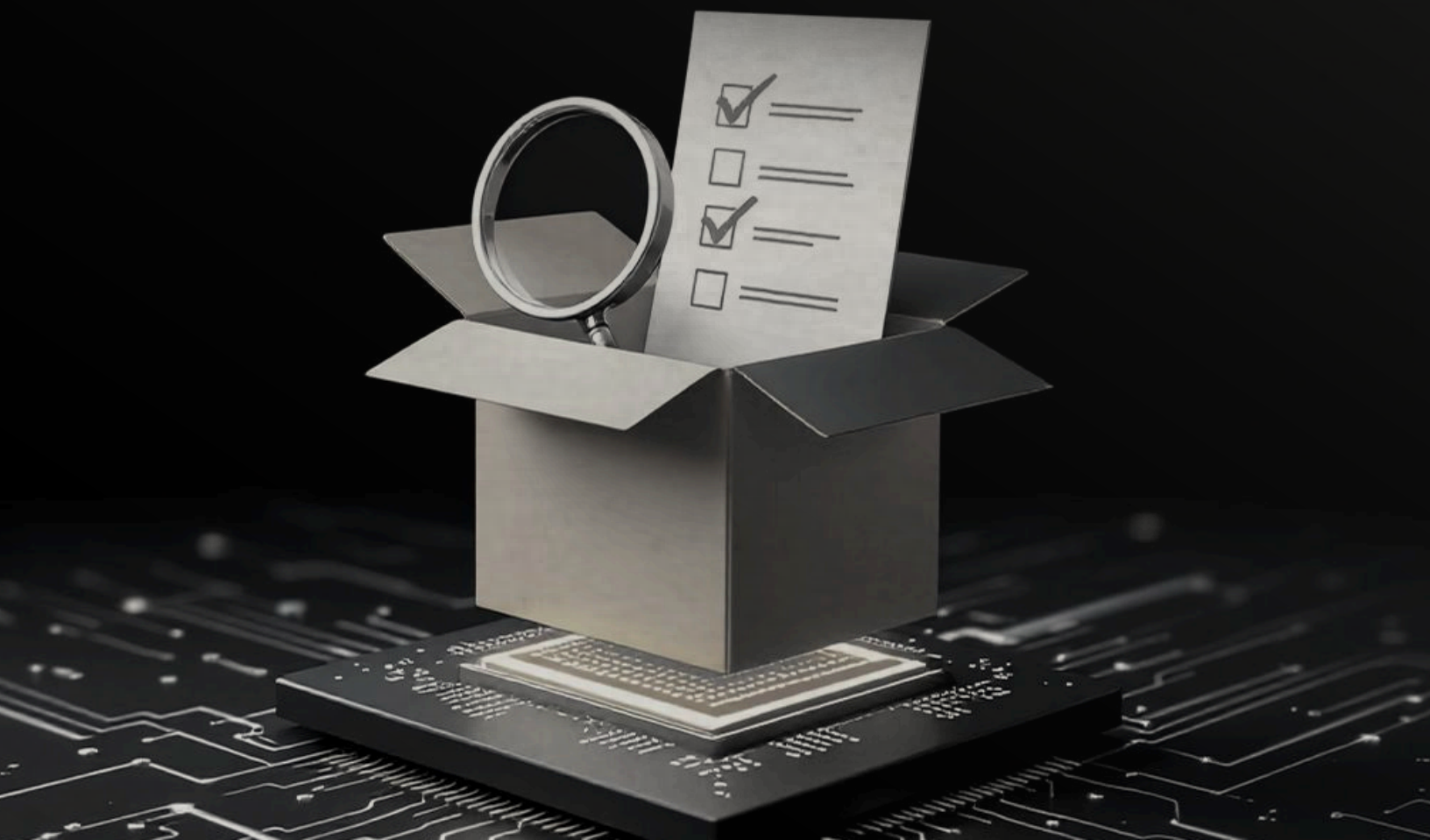
CHECKLIST

IS YOUR AI PROMPT READY FOR PRODUCTION?



IS YOUR AI PROMPT READY FOR PRODUCTION?

Deploying an AI prompt to production without proper validation is the digital equivalent of launching code without testing. This checklist consolidates the reliability engineering principles into a systematic validation process that prevents the "80% works, 20% catastrophic failure" trap.

Checklist

1. **Prompt Architecture & Structure**
    - ☐ **Implement strict input containment with delimiters**
      Wrap all user-provided data in XML tags (``, ``) or triple quotes to prevent prompt injection attacks. This creates a security boundary between instructions (logic) and variable content (data), preventing malicious or accidental instruction override. Without this, users can inject "ignore previous instructions" commands that compromise your system.
    - ☐ **Separate system instructions from user queries**
      Place all static rules, personas, output schemas, and policies in the System Prompt, not the User Prompt. This architectural separation makes instructions more resistant to injection attacks and enables prompt caching,

which can reduce costs by up to 90% and latency by 85% for repeated static content.

- ☐ **Structure prompts for cache optimization**
  Position your heaviest, unchanging content (reference documents, policies, long manuals) at the very beginning of your System Prompt. Place variable elements last. Prompt caching requires byte-perfect matches, so even a single changed character breaks the cache and forces expensive re-processing.

2. **Reasoning & Logic Validation**

- ☐ **Force explicit reasoning chains for complex tasks**
  For non-reasoning models, implement Chain of Thought prompting by instructing the model to show its work step-by-step before delivering final answers. For reasoning models (like o1), switch to outcome-based prompting that defines success criteria rather than prescribing thinking steps, as these models perform CoT internally.
- ☐ **Decompose complex workflows into atomic prompt chains**
  Break multi-step tasks (analyze, compare, rewrite) into separate sequential prompts where each handles one function. A customer support pipeline should have distinct prompts for triage, diagnosis, and drafting rather than one mega-prompt. This isolation enables precise debugging and prevents attention mechanism overload.

- ☐ **Implement "silent reasoning" for structured outputs**
  When requiring strict JSON or XML output, include a dedicated field (like `"thought_process"` or `` tags) where the model can generate its Chain of Thought. Your parsing logic extracts only the final answer field, allowing the model to think without polluting production data.

3. **Output Format & Syntax Control**
   - ☐ **Define explicit output schemas with data types**
     Specify exact JSON structures with key names and data types, or XML tag hierarchies. Don't ask for "a list"—demand `{"items": ["string", "string"], "count": int}`. Vague format requests lead to inconsistent outputs that break parsing logic.
   - ☐ **Enforce negative constraints with specific prohibitions**
     Use concrete "Do NOT" statements rather than generic requests. Instead of "don't add filler," write: "Do not output markdown code blocks. Do not output introductory text. Do not output concluding remarks." Models struggle with abstract negatives but respond to explicit prohibitions.
   - ☐ **Validate structural compliance programmatically**
     Before evaluating content quality, run automated syntax checks: Does the JSON parse? Are all required keys present? Does the XML close all tags? A response that fails structural validation should score zero and trigger immediate prompt revision.

4. **Testing & Quality Assurance**
   - ☐ **Build a Golden Dataset with 70% realistic + 30% adversarial cases**
     Curate 20-100 test cases including typical inputs, edge cases (empty strings, gibberish, extremely long text), adversarial attacks ("ignore previous instructions"), and "needle in haystack" scenarios. This dataset becomes your prompt's standardized exam that must pass before deployment.
   - ☐ **Establish measurable success metrics across three layers**
     Define passing criteria for: (1) Structural validity (parseable format), (2) Semantic accuracy (factual correctness via string matching or key fact presence), and (3) Stylistic quality (tone, brevity—use LLM-as-a-Judge for subjective criteria). Set minimum thresholds for each layer.
   - ☐ **Run full regression testing after every prompt modification**
     When fixing one failure mode, always re-test the entire Golden Dataset. The "Regression Trap" occurs when solving an edge case silently breaks previously working functionality. Version your prompts (v1.0, v1.1) and compare scores before promoting changes.

5. **Model Configuration & Tuning**

- ☐ **Set temperature based on task determinism requirements**
  Use Temperature 0.0-0.2 for data extraction, JSON generation, and coding tasks requiring zero creativity. Use 0.3-0.5 for summarization and analysis. Use 0.7-0.9 only for creative writing or brainstorming. High temperature increases hallucination risk in factual tasks.
- ☐ **Tune model-specific prompt dialects**
  Translate your instructions into the target model's preferred format: GPT-4 requires heavy negative constraints and conversational imperatives; Claude demands XML structure and "Data First, Instructions Second" flow; reasoning models (o1) need outcome-based prompting without step-by-step scaffolding that causes recursion loops.
- ☐ **Implement confidence thresholds and human-in-the-loop checkpoints**
  For high-stakes actions (financial transactions, customer communications, legal decisions), never allow full automation. Design your system to flag low-confidence responses (below 90-95%) for human review. A well-architected agent tees up decisions; humans make final calls on critical paths.

6. Production Readiness & Monitoring

- ☐ **Establish version control and prompt library infrastructure**
  Treat prompts as code assets requiring Git-style versioning, documentation, and change logs. When models update (which happens frequently), your prompts may break silently. Maintain a prompt library that tracks which version is deployed, when it was tested, and against which model version.