

Production-Ready AI: Engineering Reliable Prompts for Software Development Teams

About Otto



Sr. Software Dev @ CODE Consulting

Professor at UH

6x USPA World Records in Powerlifting

otto@codemag.com



CODE30

YEARS



30 years of
"Helping People Build Better Software"

AI Consulting Services

CODE
PRESENTS

Check out our new Executive Briefing offer!

We can help with your AI needs

What does AI mean for you?

“Skunk Works” Projects

codemag.com/AI

codemag.com/ExecutiveBriefing



Anatomy of Effective Prompts

Beyond Context and Specificity



Analyze this customer support ticket and categorize it.

The ticket content is:

"{user_ticket_content}"

Provide the category (billing, technical, account, or general) and urgency level (low, medium, high).

Improvements:

- ✓ Specific categorization request
- ✓ Defined category options
- ✓ Urgency level assessment
- ✓ Clear output expectations

Still Missing:

- ✗ Business context and rules
- ✗ Structured output format
- ✗ Key information extraction
- ✗ Response reasoning

Structure and Format Definition



Analyze the following customer support ticket and provide a structured analysis.

Ticket Content: "{user_ticket_content}"

Provide your analysis in this exact JSON format:

```
{  
  "category": "billing|technical|account|general",  
  "urgency": "low|medium|high",  
  "sentiment": "positive|neutral|negative",  
  "key_issues": ["list", "of", "main", "issues"],  
  "suggested_response_time": "1h|4h|24h|72h"  
}
```

Improvements:

- ✓ Structured JSON output format
- ✓ Specific field definitions
- ✓ Added sentiment analysis
- ✓ Key issues extraction
- ✓ Response time suggestion

Still Missing:

- ✗ Business context and rules
- ✗ Decision reasoning
- ✗ Edge case handling
- ✗ Quality validation

Role-Based with Validation and Edge Cases



You are an expert customer support analyst for TechFlow SaaS, a B2B project management platform with 50,000+ users. You have 5 years of experience in customer service operations and understand both technical and business contexts.

Your Task: Analyze the incoming support ticket and provide actionable categorization for our support team routing system.

Ticket Content: "{user_ticket_content}"

Company Context:

- Product: Cloud-based project management software
- Customer Base: Small to enterprise businesses (50-5000 employees)
- Support Hours: 24/7 for Enterprise, business hours for Standard plans
- Common Issues: Integration problems (30%), billing questions (25%), feature requests (20%), bugs (15%), account issues (10%)

Categorization Framework:

Primary Categories:

- "billing": Payment failures, subscription changes, invoicing, refunds, plan upgrades/downgrades
- "technical": Software bugs, API issues, integration problems, performance, data sync
- "account": Login/logout, user permissions, SSO setup, account settings, user management
- "general": Feature requests, how-to questions, feedback, training requests

Urgency Matrix:

- "high": Service completely down, security breach, billing dispute >\$1000, enterprise customer angry
- "medium": Feature partially broken, integration failing, standard billing issue, frustrated tone
- "low": Feature requests, documentation questions, positive feedback, minor bugs with workarounds

Analysis Requirements:

1. Extract the customer's primary concern and any secondary issues
2. Identify emotional indicators and urgency cues
3. Determine if technical expertise is required
4. Check for enterprise customer indicators (mentions teams >100, integration, compliance)
5. Flag potential escalation needs

Output Constraints:

- Always return valid JSON with all required fields
- If uncertain about category, choose the most specific applicable option
- For mixed issues, categorize by the most urgent/impactful problem
- Include actual quoted text from ticket in key_issues where relevant
- Ensure suggested_response_time aligns with urgency level

Validation Rules:

- key_issues must contain 1-3 specific, actionable items
- reasoning must explain the categorization logic
- confidence_score must reflect certainty level (0.1-1.0)

Required JSON Output:

```
{
  "category": "billing|technical|account|general",
  "urgency": "low|medium|high",
  "sentiment": "positive|neutral|negative",
  "customer_emotion": "calm|frustrated|angry|satisfied|confused",
  "key_issues": ["specific issue 1", "specific issue 2"],
  "technical_expertise_required": true|false,
  "enterprise_customer_likely": true|false,
  "suggested_response_time": "1h|4h|24h|72h",
  "escalation_needed": true|false,
  "reasoning": "Brief explanation of categorization logic",
  "confidence_score": 0.85
}
```

Edge Case Handling:

- Empty or very short tickets: Mark as "general" with "low" urgency
- Aggressive/threatening language: Always mark escalation_needed as true
- Multiple unrelated issues: Focus on the most urgent one
- Unclear technical terms: Don't assume high urgency without clear impact indicators

```
{
  "category": "technical",
  "urgency": "high",
  "sentiment": "negative",
  "customer_emotion": "angry",
  "key_issues": [
    "Entire team unable to access the dashboard",
    "Loss of productivity due to downtime"
  ],
  "technical_expertise_required": true,
  "enterprise_customer_likely": true,
  "suggested_response_time": "1h",
  "escalation_needed": true,
  "reasoning":
    "The ticket indicates a complete service outage
    ('Our entire team can't access the dashboard since this morning'),
    which aligns with the 'technical' category. The customer emphasizes
    the financial impact ('costing us thousands in lost productivity'),
    suggesting an enterprise customer. The tone indicates anger and
    frustration due to the severity of the issue. Given the total service
    disruption and potential enterprise-level customer, urgency is 'high'
    and escalation is required to resolve the issue promptly.",
  "confidence_score": 0.95
}
```

Problem Example: .NET 5,6,7,8,9,10



- Overall original content winner:
.NET 5 – the entire unified platform
- Close second: .NET 6, first LTS release
- **Problem:** updated/not updated content
- ➔ ALWAYS BE SPECIFIC



Advanced Prompt Optimization Techniques

Code Reviews



Problem: Code review this function / snippet

Base Output: A wall of text of everything that's good and bad

Improved Simple Prompt: Is there anything bad or wrong with my code? Be short and precise!

Improved Complex Prompt: Is there anything bad or wrong with this code? Do not list approaches that are well done, instead categorize the issues you see in three categories:

- 1) must fix (typos, obvious bugs, logic mistakes, wrong data types)
- 2) potential fixes (could be fixed or improved by a different approach, might or might not be a problem, depending on code not shown, could pose a problem later on)
- 3) optimization (things that work as they are, do not need to be fixed, but could be improved using a different approach and/or algorithm)

logit bias -100



The API generates tokens one by one.

Each possible token has a probability (logit $\rightarrow$ softmax $\rightarrow$ probability).

Logit bias lets you nudge or block specific tokens:

A **positive bias** makes a token more likely.

A **negative bias** makes a token less likely.

A bias of **-100** basically *blocks* that token (probability ≈ 0).

JSON payload example for API call



```
{  
  "model": "gpt-4o-mini",  
  "prompt": "Give me a list of tech companies.",  
  "logit_bias": {  
    "18900": -100, // token ID for "Google"  
    "25667": -100  // token ID for "Microsoft"  
  }  
}
```

CrowdStrike → [124721, 67, 109630] → CrowdStrike

Finding Tokens (python)



```
import tiktoken  
enc = tiktoken.encoding_for_model("gpt-4o-mini")  
print(enc.encode("Google"))
```

<https://platform.openai.com/tokenizer>

Remove comments

Ban “//” or “#”:

Prohibit comments entirely, forcing the model to write everything in code, rather than leaving blocks with:

//to be implemented

#implement sort algorithm here

top-k parameter

When the model generates the next token, it calculates a probability distribution over the entire vocabulary, often tens of thousands of tokens.

Top-k sampling means:

1. Sort tokens by probability.
2. Keep only the **k most likely tokens**.
3. Renormalize their probabilities.
4. Randomly pick one of them as the output.
5. All other tokens outside the top-k set are ignored.

top-k

```
{  
  "model": "gpt-4o-mini",  
  "prompt": "Write a haiku about the ocean.",  
  "top_k": 50  
}
```

top_k = 1: Greedy decoding — the model always picks the single most likely next token.
→ Output is deterministic but can feel repetitive or dull. (coding)

top_k = 10: Limits randomness to the top 10 tokens.
→ Balanced: coherent but with some diversity. (natural writing)

top_k = 100: The model considers 100 options.
→ More creative, but risk of going off-topic or rambling. (brainstorming, poetry)

Structured Reasoning Frameworks



1. Decompose
2. Hypothesize
3. Criticize
4. Synthesize
5. Reflect

Decompose



Instead of throwing one big vague prompt, you break it into *smaller, clear pieces*.

This reduces ambiguity and makes the model (or yourself) focus on distinct elements.

Example: Instead of asking “*How can I improve my startup?*” you split into “*How can I reduce costs?*”, “*How can I improve customer acquisition?*”, etc.

Hypothesize



By prompting for *multiple possible answers* per sub-question, you get diversity instead of a single “default” response.

This prevents tunnel vision and encourages creativity in outputs.

Example: “*List at least 3 different approaches to reducing costs.*”

Criticize

- Adding a step where you or the model *evaluate weaknesses* stops you from blindly accepting the first good-sounding idea.
- In prompting, you can explicitly ask the model to “play devil’s advocate” or “list flaws in the proposed solution.”

Synthesize



You then guide the model to *merge the best parts of the options*.

This avoids the weakness of “list mode” answers that don’t converge into a recommendation.

Example: *“Take the two most feasible cost-saving methods and combine them into a single, optimized plan.”*

Reflect

Finally, asking the model to *explain its reasoning, highlight uncertainties, and suggest next steps* helps avoid overconfidence and shows gaps.

This makes the answer more actionable because it doesn't just give "what to do" but also "what still needs checking."

Outcome



It gives the model a **scaffold** for reasoning, rather than hoping it will reason well unstructured.

It reduces hallucination risk
(because you force critique and reflection).

It increases creativity + reliability
(because you generate, critique, then refine).
It's closer to how experts solve problems —
iterative, structured, self-critical.



Image Prompts

Create an image of a Fruit basket



Create an image of a fruit basket with asserted fruits.

VS

```
{
  "prompt": "A photorealistic image of a rustic wicker basket filled with fresh, vibrant fruits including apples, oranges, strawberries, and bananas, with delicate watery mist droplets clinging to the fruit surfaces, capturing the freshness. The scene is set on a natural wooden table with soft, diffused natural light streaming from a nearby window, creating gentle shadows and highlights. The textures of the wicker, fruit skins, and water droplets are highly detailed and tactile, rendered in 8K resolution for crisp clarity and realism, suitable for a high-end commercial advertisement.",
  "parameters": {
```

```
    "style": "photorealistic",
    "resolution": "7680x4320",
    "aspect_ratio": "3:2",
    "camera_settings": {
      "camera_type": "DSLR",
      "lens": "50mm prime",
      "aperture": "f/2.8",
      "shutter_speed": "1/125s",
      "ISO": 100,
      "focal_length": "50mm",
      "focus_mode": "manual",
      "depth_of_field": "shallow",
      "white_balance": "daylight"
    },
```

```
    "lighting": {
      "type": "soft natural light",
      "source": "window light from left",
      "intensity": "medium",
      "shadows": "soft",
      "highlights": "gentle"
    },
    "texture_details": {
      "wicker_basket": "rough, woven, high detail",
      "fruit_surfaces": "glossy with micro-textures, visible pores and imperfections",
      "water_droplets": "translucent, reflective, high clarity"
    },
    "color_settings": {
      "color_grading": "natural, vibrant",
      "saturation": "high but realistic",
      "contrast": "moderate",
      "brightness": "balanced"
    },
```

```
    "render_settings": {
      "anti_aliasing": "16x",
      "texture_resolution": "8K",
      "ray_tracing": "enabled",
      "global_illumination": "high",
      "reflections": "realistic, subtle"
    },
    "post_processing": {
      "sharpening": "slight",
      "noise_reduction": "minimal",
      "lens_flare": "subtle",
      "vignette": "light"
    },
    "background": {
      "setting": "indoor, rustic kitchen",
      "elements": "blurred wooden table, soft-focus window in background",
      "depth": "shallow to keep focus on basket"
    },
    "commercial_use": {
      "purpose": "high-end food advertisement",
      "clarity_requirement": "sharp, professional, print-ready",
      "color_accuracy": "true-to-life for fruit vibrancy"
    }
  }
}
```

Gemini

CODE
PRESENTS



Grok

CODE
PRESENTS



Table Setting



A notoriously difficult problem:

Generate an image of a formally
set table for 6 persons

Gemini

CODE
PRESENTS



- Weird candles
- 5 instead of 6 persons
- Mis-matching number of pieces on the right side (2v3)
- 1 set of wine glasses missing
- Inconsistent napkin decorations

GPT 5

- 5 instead of 6 persons
- Weird positioning
- Inconsistent left silverware
- Missing glass
- Off center flowers
- Missing napkins
- Missing plate



Grok

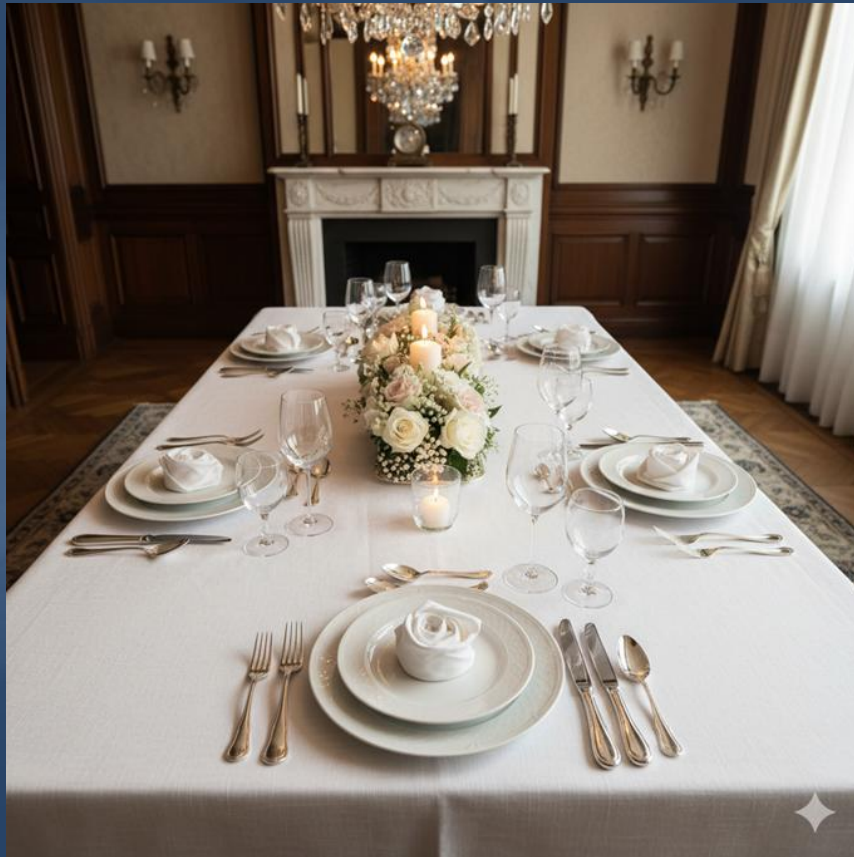
CODE
PRESENTS



- Created for 8 persons
- Most glasses missing
- Inconsistent dessert silverware
- Missing small bread plate plus silverware

Optimized Prompts:

CODE
PRESENTS



Turbocharge your prompts:



https://github.com/asgeirtj/system_prompts_leaks/blob/main/OpenAI/tool-create-image-image_gen.md

Lists expected prompt format, installed frameworks and tools/libraries, etc., e.g. for GPT 5.2:

Environment

- `reportlab` is installed for PDF creation. You *must* read `/home/oai/skills/pdfs/skill.md` for tooling and workflow instructions.
- `python-docx` is installed for document editing and creation. You *must* read `/home/oai/skills/docs/skill.md` for tooling and workflow instructions.
- `pptxgenjs` is installed for slide creation. Image tools and JS helpers are available at `/home/oai/share/slides/`.
- `artifact_tool` and `openpyxl` are installed for spreadsheet tasks. You *must* read `/home/oai/skills/spreadsheets/skill.md` for important instructions and style guidelines.

ONE CODE



CODE Staffing

CODE
PRESENTS

Disrupting the world of staffing!

Giving our customers the ability to have staff on par with Silicon Valley companies ...

... and our employees a work environment in a bleeding-edge tech company with the **industry leading benefits!**

codestaffing.com





Subscribe and “ring that bell”
to never miss any of our content!

youtube.com/codemag



CODE Magazine

@Codemag 2.91K subscribers 69 videos

For over 20 years, CODE Magazine has provided technical content in our pr... >

Subscribed ▾

Free Subscription

CODE
PRESENTS

The leading software development magazine,
written by expert developers for developers.

All registered attendees will receive a
free digital subscription to CODE Magazine!

Share this link to our free subscription:
<https://bit.ly/CP-01-14-26Subscribe>



Q&A

CODE
PRESENTS

Contact us with questions!



CODE Contact

codemag.com
info@codemag.com
facebook.com/codemag
twitter.com/codemagazine

Presenter Contact

otto@codemag.com