Here are several prompt-engineering strategies you can apply to reduce hallucinations when asking the model for a technical‐analysis JSON output:

1. **Strengthen your system prompt with explicit “don’t make things up” instructions**

Instead of simply stating “Provide accurate, data-driven predictions,” add a clause like:  
“You are a professional financial analyst… Rely strictly on the data provided. If you don’t have enough information to compute something, respond with a null or an ‘insufficient data’ flag rather than invent numbers. Do not fabricate prices, dates, or volumes. Always respond with valid JSON only.”

* + This extra “if-insufficient-data, reply null” language makes it clear the model should refuse to hallucinate details that aren’t in the 78 data points you attached.

1. **Lower temperature even further (to 0.0 or 0.1) for maximal determinism**
   * You currently have "temperature": 0.3. Dropping that to "temperature": 0.0 (or at most 0.1) makes the model far less likely to conjure “plausible-sounding” but made-up numbers.

Example:  
"temperature": 0.0

* + With zero temperature, you trade off some flexibility in phrasing, but you significantly reduce “creative” outputs.

1. **Tell the model how you want it to handle missing or ambiguous data**

Add a short instruction like:  
“If any support/resistance or confidence interval cannot be computed from the provided 78 five-minute data points, output `"null"` for that field or include `"reason": "insufficient data to calculate"`. Never invent price levels or volumes that aren’t explicitly given.”

* + That makes the fallback behavior explicit rather than allowing the model to hallucinate a “typical” moving average or RSI.

1. **Break complex analyses into sub-steps within the JSON schema**

Instead of one big freeform request, specify sub-objects for each part of the output. For example:  
{

"model": "gpt-4o",

"messages": [

{

"role": "system",

"content": "You are a professional financial analyst… [“no fabrication” instruction as above]."

},

{

"role": "user",

"content": "Here are 78 data points (timestamp, price, volume).

Please return precisely these JSON root keys:

{

\"predictions\": {

\"1d\": {\"point\": number, \"low\": number or null, \"high\": number or null, \"confidence\": number},

\"1w\": {…},

\"1m\": {…}

},

\"technical\": {

\"trend\": \"up\" | \"down\" | \"sideways\" | null,

\"support\_levels\": [number, …],

\"resistance\_levels\": [number, …]

},

\"reasoning\": { … },

\"recommendation\": \"buy\" | \"sell\" | \"hold\" | null

}

Use only the provided 5-minute data to calculate moving averages, trend lines, support/resistance. If you cannot compute a field exactly, set it to null and put a brief “reason”: “insufficient data.””

}

],

"temperature": 0.0,

"response\_format": { "type": "json\_object" }

}

* + Defining the exact JSON schema up front (with typing and null allowances) forces the model to fit into your template rather than free-text hallucinations.

1. **Supply any formulas or indicator definitions explicitly**

If you want (say) a 20-period simple moving average (SMA) or RSI, tell the model exactly how to compute it:  
“To compute a 20-period SMA: sum the last 20 closing prices and divide by 20.

To compute RSI over 14 periods: [brief formula in words].

Use only those definitions when you calculate.”

* + By giving it the formulas, you reduce “model approximation” of indicator calculations and keep it strictly data-driven.

1. **Ask for a short “data‐validation” step first**

You can prepend a request like:  
“First, please check that the input JSON contains exactly 78 time‐stamped price‐volume entries at 5-minute intervals. If any entry is missing or malformed, return { \"status\": \"error\", \"message\": \"Please supply valid 78 entries.\" }. Otherwise, proceed to analysis.”

* + That way, if your website accidentally sends malformed or truncated data, the model refuses rather than hallucinating.

1. **Use function calling (if available) for numeric sub-tasks**

If your environment supports OpenAI functions, you can define a function like:  
{

"name": "calculate\_moving\_average",

"description": "Compute the simple moving average for given price list and period",

"parameters": {

"type": "object",

"properties": {

"prices": {

"type": "array",

"items": { "type": "number" },

"description": "Array of closing prices"

},

"period": { "type": "integer", "description": "Number of points to average" }

},

"required": ["prices","period"]

}

}

* + Then call the function for each SMA or indicator rather than letting the LLM do freeform math. The model will return { "name": "calculate\_moving\_average", "arguments": { … } }. Your code runs that calculation exactly and feeds the result back to the assistant. This eliminates risk of the model inventing an SMA value.

1. **Include “conflict‐check” instructions**

Add a line such as:  
“Cross-check each support/resistance level against the raw data. If the proposed level is more than 0.5% away from any data point in the visible range, mark it invalid.”

* + That encourages the model to literally “look back” at the array of prices instead of guessing common pivot zones.

1. **Enforce a motto of “If it’s not explicitly derivable, output null”**

Repeat in both system and user prompts:  
“If any value cannot be calculated to at least two decimal places from the provided data, output null and do not fabricate.”

* + For example, if you don’t give them enough history to calculate a 50-day moving average, they must return null rather than guess a plausible “next month” price.

1. **Example JSON template with placeholders**

Finally, include a one‐entry “example response” so the model sees the exact format you expect. E.g.:  
{

"predictions": {

"1d": {

"point": 201.23,

"low": 198.50,

"high": 203.10,

"confidence": 65

},

"1w": {

"point": 205.00,

"low": null,

"high": null,

"confidence": 50

},

"1m": {

"point": null,

"low": null,

"high": null,

"confidence": 0

}

},

"technical": {

"trend": "up",

"support\_levels": [200.00, 198.75],

"resistance\_levels": [203.50],

"reason\_for\_trend": "Price has made a higher high every four hours over the past 12 hours"

},

"recommendation": "buy"

}

* + Label this snippet as an “Example of correct JSON structure.” That anchors the model’s output shape and reduces freeform commentary.

### **Putting It All Together**

A revised JSON payload might look like this:

{

"model": "gpt-4o",

"messages": [

{

"role": "system",

"content": "You are a professional financial analyst with expertise in technical analysis and stock prediction. Rely strictly on the data provided. If you cannot derive a metric from those 78 five-minute entries, return null for that field and include a brief “reason”. Do not fabricate any numbers. Always respond with valid JSON only."

},

{

"role": "user",

"content": "Below are exactly 78 five-minute data points for AAPL.

Historical Price Data (last 78 points):

[ {\"time\": \"13:30\", \"price\": 202.70, \"volume\": 2433892},

{\"time\": \"13:35\", \"price\": 202.65, \"volume\": 2435124},

… (76 more) … ]

Please output exactly this JSON schema (no additional keys):

{

\"predictions\": {

\"1d\": {\"point\": number or null, \"low\": number or null, \"high\": number or null, \"confidence\": integer 0–100},

\"1w\": {same structure},

\"1m\": {same structure}

},

\"technical\": {

\"trend\": \"up\"|\"down\"|\"sideways\"|null,

\"support\_levels\": [number, …] or [],

\"resistance\_levels\": [number, …] or [],

\"reason\_for\_trend\": string or null

},

\"recommendation\": \"buy\"|\"sell\"|\"hold\"|null

}

Use these definitions:

• 20-period SMA = average of last 20 closing prices.

• 14-period RSI = 100 – (100 / (1 + (avg\_gain/avg\_loss))) over last 14 closes.

• A trend is “up” if the 20-period SMA is rising over the last 4 intervals by at least $0.10; “down” if falling similarly; otherwise “sideways”.

• Support = any local minima in price where price rebounded at least twice within ±0.5%.

• Resistance = any local maxima meeting the same ±0.5% rebound criteria.

• For predictions: extrapolate only from a linear regression on the last 50 five-minute closes. If the R² of that regression is below 0.2, set all prediction fields to null and confidence to 0.

• Confidence = integer 0–100 proportional to R² (e.g., R²=0.8 → 80).

If you cannot compute a field exactly as defined, output null (not 0) and set confidence to 0. Respond ONLY with the JSON object—no surrounding text or extra punctuation."

}

],

"temperature": 0.0,

"response\_format": { "type": "json\_object" }

}

#### **Why this helps:**

* **Explicit “no fabrication” rules** in the system prompt and user prompt force the model to refuse to invent.
* **Clear JSON schema + example** anchor the shape of the output.
* **Zero temperature** makes outputs deterministic.
* **Detailed indicator formulas** prevent the model from guessing indicator values.
* **R² threshold rule** means the model can refuse if the data doesn’t support a confident prediction, rather than hallucinating confidence.
* **“If you cannot calculate exactly, output null”** removes incentives to guess.

With these changes, the model must (1) use only your 78 data points, (2) follow your formulas, and (3) output exactly the JSON shape. If any piece of information is missing, it will return null instead of making up numbers—drastically reducing hallucination risk.