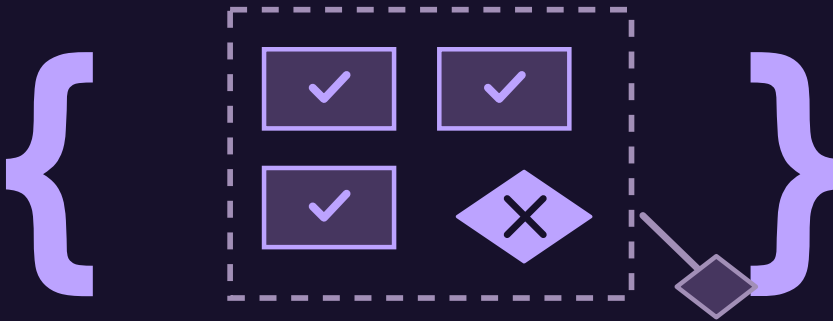




PRACTICAL AI

Validate Structured AI Outputs

Turn plausible JSON into a reviewable data contract.



Sanjaay Gowda T S

FREE SAMPLE / INTRODUCTION + CHAPTER 1 / EDITION 1.0.0

sanjaaygowdats.ai

About this free sample

Validate Structured AI Outputs

Sanjaay Gowda T S

First edition 1.0.0, 2026-10-13

Copyright 2026 Sanjaay Gowda T S. This is a labelled free sample, not the complete ebook. You may share this sample unchanged. It contains the cover, sample permissions, contents, introduction, Chapter 1 and a closing product link. The remaining chapters, worked solutions and companion files are not included. The full ebook text, diagrams and artwork have separate single-reader terms; this sample permission does not authorize sharing the full purchased PDF or EPUB.

The companion source code, tests and fictional data are separately MIT licensed. You may adapt or share that project with its included licence notice. MIT does not apply to the ebook text or book artwork. The books and companion projects have no technical activation requirement or buyer-specific watermark.

This first edition was prepared with AI assistance. The case studies are fictional teaching examples. Any measured outputs come from the supplied examples, not production customer data. The teaching scope, assumptions and limitations are explained in the chapters. Independent human technical, editorial and assistive-technology reader review is pending. Automated tests and accessibility checks do not establish certification or suitability for every use.

Run examples in a separate working folder using the supplied synthetic data. Review inputs and calculations before adapting the techniques to real operations. Model software, service access, subscriptions, ongoing updates and consulting are not included.

Author website: sanjaaygowdats.ai

Ebook terms: sanjaaygowdats.ai/ebooks/license

Inside this free sample

Start here.....	4
1. Specify the output contract.....	5
Continue with the full guide	7

This free sample contains the introduction and Chapter 1 only. The closing link opens the complete guide. The companion project and worked solutions are supplied with the complete ebook.

Start here

A generated response can be valid JSON and still be unusable data. It may omit a required field, encode a quantity as a sentence, repeat an object key, attach a quote that never appeared in the source, or interpret a negated request as an instruction to order something. Each failure belongs at a different boundary. This book helps you design those boundaries so that an application can reject uncertain data without hiding why.

You will build a validator for fictional stationery requests. The extraction contract has a decision, item, quantity, priority and evidence. Every source message and output in the companion is a synthetic, hand-written fixture. They were not produced by a language model, customer or extraction service. The project calls no model and performs no purchase, email or other external action. Its job is to make the acceptance rules inspectable.

Use Python 3.10 or later, with basic dictionaries, exceptions and JSON familiarity. Extract the project, open a terminal where `main.py` and `example-input.json` live, and run:

SHELL

```
python --version
python main.py --input example-input.json
python -m unittest discover -s tests -v
```

On Windows, `py -3` can replace `python`. On macOS or Linux, `python3` may be the installed command. No package installation or API key is required. Compare the resulting JSON with `expected-output.json`. The tests cover deliberate failures as well as accepted outputs. A passing test suite establishes these local behaviors; it does not certify any real model's extraction quality.

The outcome is a data contract with explicit refusal and error handling. The companion checks strict parsing, exact fields, selected business rules and shallow text correspondence. Semantic interpretation remains a separate review obligation. A supplied test deliberately shows an accepted interpretation that reverses a negated message, making that limitation concrete rather than leaving it as a vague disclaimer.

1. Specify the output contract

Describe the consumer's need

Imagine a fictional internal desk that receives stationery requests. A staff member wants a draft record of the requested item, quantity and declared priority. The draft will be reviewed before any action. Define these needs before choosing a JSON shape. If the consumer needs an exact count, accepting "a few" as a number changes the workflow rather than merely loosening validation.

The teaching contract uses two decisions: extract and abstain. Extract requires a nonempty item string, an integer quantity from one through one hundred, a priority equal to standard or urgent, and evidence quotes for all three fields. Abstain requires null item, quantity and priority plus an empty evidence object. This design keeps a partial guess from masquerading as a complete request.

Field	Extraction requirement	Abstention requirement
decision	extract	abstain
item	Nonempty bounded string	null
quantity	Integer 1-100	null
priority	standard or urgent	null
evidence	Three named source quotes	Empty object

These limits are invented for the exercise. They are not recommendations about real ordering quantities or business urgency. A real workflow needs its own policy owner. The important technique is to make the accepted domain explicit and to distinguish missing information from values such as zero, an empty string or a default priority.

Decide what additional fields mean

The validator accepts exactly five top-level fields. An extra field is rejected. This prevents an upstream change from slipping through unnoticed, but it means contract migrations need planning. If a future producer adds `requested_date`, the old consumer should fail visibly until both sides agree how to handle it. Silently ignoring additions can conceal a misunderstanding about which data is authoritative.

JSON Schema provides concepts for required fields and additional properties [R2]. The companion implements its own narrow rules directly in Python; it is not a JSON Schema engine. Do not copy its bespoke functions and advertise general schema support. Learning the concepts and implementing an entire specification are different scopes.

State acceptance precisely

Accepted extraction means the shape and specified text checks passed. It does not mean the source describes a genuine request, that the sender is authorized, that the item exists, or that a purchase is approved. Keep acceptance names close to the actual guarantees. A field named `verified_order` would overstate what this validator establishes.

Exercise: write the consumer contract

Write a paragraph describing who reviews the extracted record, why exact quantity matters and when abstention is required. Add one valid extraction and one valid abstention as full JSON objects. Explain why a missing quantity must not become zero or one merely to satisfy a required field.

Continue with the full guide

This free sample ends after the introduction and Chapter 1. The complete guide contains nine teaching chapters, worked exercise solutions, primary references and a separate Python companion project. The full PDF, reflowable EPUB and project files are supplied together.

Use the complete guide to work through the remaining exercises and inspect the supplied examples. The companion code and fictional data are separately MIT licensed; the complete ebook has the single-reader terms described on its product page.

The examples are teaching fixtures, not production customer results. Independent human technical, editorial and assistive-technology reader review remains pending. Software access, subscriptions, consulting and guaranteed results are not included.

[View the complete guide and purchase options.](#)

You may share this labelled free sample unchanged. Keep the full purchased PDF and EPUB private. Thank you for reading.

Primary references

1. [R2] JSON Schema object reference

<https://json-schema.org/understanding-json-schema/reference/object>

Required properties and additionalProperties concepts; companion is a bespoke validator, not a schema implementation.