das-purnendu

PYTHON SKILLS

EVERY GENAI ENGINEER MUST KNOW

2026 HANDBOOK

A practical field guide for building reliable LLM apps with clean Python, typed schemas, async APIs, RAG, evals, observability, agents, and production guardrails.

Build

typed, testable model workflows

Retrieve

evidence with citations and filters

Ship

traces, evals, budgets, and safety

Keep this nearby for **agents**, **RAG**, and **AI product backends**.

Save this handbook for your **2026 GenAI** builds →

das-purnendu

GENAI PYTHON SKILL MAP

The stack you actually touch

Product API
FastAPI, CLI, jobs

→ **Orchestration**
chains, agents, queues

→ **Model Clients**
timeouts, retries

→ **Tools + RAG**
schemas, retrieval

→ **Eval + Ops**
tests, traces

Python core

- [x] types
- [x] dataclasses
- [x] iterators
- [x] context managers

Data layer

- [x] JSON
- [x] CSV / Parquet
- [x] Pydantic
- [x] Pandas / Polars

Network layer

- [x] httpx
- [x] asyncio
- [x] backoff
- [x] streaming

AI layer

- [x] prompts
- [x] tool calls
- [x] embeddings
- [x] evaluations

Great GenAI engineers are not prompt-only engineers.
They write **boring, typed Python** around unpredictable model behavior.

Save this handbook for your 2026 GenAI builds →

das-purnendu

ENV SETUP THAT AGES WELL

terminal

```
$ uv init genai-app
$ cd genai-app
$ uv add pydantic pydantic-settings
$ uv add httpx tenacity rich pytest
$ uv run python -m pytest
$ uv lock
```

2026 baseline

- Prefer Python 3.12+ for modern typing and speed.
- Python 3.14 is current stable in 2026; test library support before prod.
- Pin dependencies and keep a lock file in git.
- Use .env locally; real secrets live in a vault.

Project shape

src/app/	business logic, prompts, tools
src/app/clients/	model, vector DB, storage adapters
tests/	unit, eval, replay fixtures
configs/	safe defaults, no secrets

Install once, reproduce everywhere

If a teammate cannot run your eval suite from a fresh checkout, your AI app is still a notebook experiment.

Save this handbook for your 2026 GenAI builds

das-purnendu

TYPE HINTS SAVE AGENTS

typing

```python
from typing import Literal, Protocol, TypedDict

Role = Literal["system", "user", "assistant", "tool"]

class Message(TypedDict):
    role: Role
    content: str

class LLM(Protocol):
    async def chat(self, messages: list[Message]) -> str: ...
```

Use	When it helps	GenAI payoff
Literal	small allowed values	role, mode, provider names
TypedDict	plain JSON payloads	clear API contracts
Protocol	swappable clients	test with fake model clients
TypeAlias	reused complex shapes	readable pipeline signatures

Must know

Typing is not bureaucracy. It is how you keep model messages, tool inputs, and retrieval metadata from becoming mystery dictionaries.

Save this handbook for your 2026 GenAI builds

das-purnendu

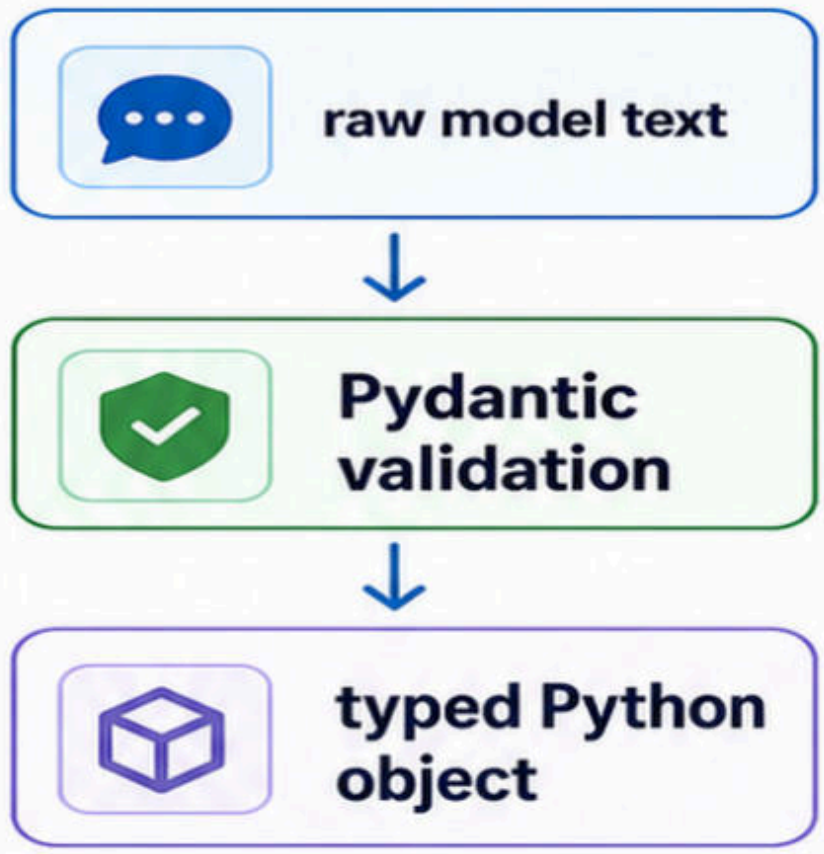
PYDANTIC MODELS FOR AI IO

{ } schema

```python
from pydantic import BaseModel, Field

class Answer(BaseModel):
    summary: str
    confidence: float = Field(ge=0, le=1)
    citations: list[str]

result = Answer.model_validate_json(raw)
```



Validate at every boundary

- User input before tool execution
- Model output before database writes
- Retrieved metadata before context packing
- Webhook payloads before async jobs

Trap

Do not silently coerce nonsense.
Return a repair prompt, a validation error, or a safe fallback.

Save this handbook for your 2026 GenAI builds →

das-purnendu

ASYNC PYTHON FOR LLM APPS

concurrency

```python
import asyncio

sem = asyncio.Semaphore(8)

async def embed(doc):
    async with sem:
        return await client.embed(doc)

async def main():
    vectors = await asyncio.gather(
        *(embed(d) for d in docs)
    )
    return vectors

vectors = asyncio.run(main())
```

Async checklist

- [x] Use timeouts around every network call.
- [x] Limit concurrency with Semaphore.
- [x] Cancel jobs when users disconnect.
- [x] Separate CPU work from IO work.
- [x] Keep sync SDK calls out of the event loop.

Mental model

await
pause this task

→

event loop
run other IO

→

resume
collect result

Production default:
async IO, bounded parallelism, explicit timeouts.

Save this handbook for your 2026 GenAI builds

das-purnendu

HTTP CLIENTS LIKE AN ADULT

httpx

```python
import httpx

timeout = httpx.Timeout(30, connect=5)

async with httpx.AsyncClient(
    timeout=timeout,
    headers={"Authorization": f"Bearer {token}"},
) as http:
    r = await http.post(url, json=payload)
    r.raise_for_status()
```

Use f"Bearer {token}" for Authorization

For OpenAI, Anthropic, and most providers this needs to be f"Bearer {token}". Readers copy-pasting will get 401s and wonder why.

Never skip

- [x] connect and read timeouts
- [x] retry only safe failures
- [x] log request IDs, not secrets
- [x] idempotency key for writes
- [x] provider-specific error mapping

Failure	Handle it	Why
429	backoff + jitter	avoid retry storms
5xx	bounded retry	transient provider issue
timeout	retry or queue	user waits less
4xx	fix request	bad schema or auth

Good clients turn flaky networks into predictable product behavior.

Save this handbook for your 2026 GenAI builds

das-purnendu

TOKENS + COST CONTROL LOOP

Budget formula

```
cost = input_tokens * in_rate + output_tokens * out_rate
```

💡 Track it per request, per user, per feature.

Before call

- [x] count tokens
- [x] trim history
- [x] cap context

During call

- [x] stream output
- [x] set max tokens
- [x] cancel on disconnect

After call

- [x] store usage
- [x] cache result
- [x] watch anomalies

guardrail

```python
if usage.total_tokens > budget.max_tokens:
    raise BudgetExceeded(feature="chat")

cache_key = hash_prompt(prompt, model, schema_version)

metrics.count("llm.tokens", usage.total_tokens)
```

Save this handbook for your 2026 GenAI builds →

das-purnendu

PROMPTS AS CODE

Prompt file rule

- One task per prompt.
- Inputs are named variables.
- Output shape is explicit.
- Dangerous instructions are isolated.
- Version prompts like code.
- Snapshot expected behavior.

prompt.py

```
PROMPT_ID = "qa.answer.v3"

template = """
Use only the supplied context.
Question: {question}
Context:
{context}
Return JSON matching Answer.
"""
```

Injection note

Retrieved text is data, not instructions. Wrap it in a clearly labeled context block and restrict tools by policy.

A good prompt has tests

Store 10-50 representative inputs. When the prompt changes, compare schema validity, citations, refusal behavior, and latency.

Save this handbook for your 2026 GenAI builds

das-purnendu

STRUCTURED OUTPUTS THAT DO NOT BREAK

output model

```python
class ExtractedTask(BaseModel):
    title: str
    owner: str | None = None
    due_date: str | None = None
    priority: Literal["low", "med", "high"]

task = ExtractedTask.model_validate_json(raw)
```

Repair loop

generate

↓

validate

↓

accept or repair

Design the schema for downstream code

- Prefer enums over free text.
- Use lists of objects, not parallel lists.
- Keep optional fields truly optional.
- Add descriptions for ambiguous fields.

Anti-pattern

Parsing prose with regex after the model returns.
Ask for a schema first, validate second, repair third.

Save this handbook for your 2026 GenAI builds

das-purnendu

TOOL CALLING WITHOUT CHAOS

tool contract

```python
class SearchArgs(BaseModel):
    query: str
    top_k: int = Field(ge=1, le=10)

async def search_docs(args: SearchArgs):
    return await retriever.search(args.query,
                  k=args.top_k)
```

Tool rules

- Validate args before work.
- Return compact, typed data.
- Separate read tools from write tools.
- Require approval for expensive side effects.
- Log tool name, latency, result size.

Tool router pattern

model
selects tool

→

validator
Pydantic args

→

executor
policy + timeout

Never let the model call arbitrary Python.
Expose a small allowlist of boring, audited functions.

Save this handbook for your 2026 GenAI builds

das-purnendu

RAG PIPELINE BLUEPRINT

Ingest
files, URLs, DB rows

→

Parse
text, tables, images

→

Chunk
semantic windows

→

Embed
vectors + metadata

↓

Index
hybrid search

←

Retrieve
filter + rerank

←

Generate
answer + citations

What makes RAG useful

- Good metadata beats bigger chunks.
- Hybrid search catches exact names and fuzzy meaning.
- Reranking improves relevance when top_k is noisy.
- Citations make answers reviewable.

Key interface

```
retrieve(query, filters) -> list[Chunk]
```

Keep retrieval testable as its own Python function.
Do not bury it inside one giant prompt.

Save this handbook for your 2026 GenAI builds

das-purnendu

CHUNKING THAT RETRIEVES

Content	Chunk by	Add metadata
Docs	headings + paragraphs	title, section, page
Code	symbol + imports	repo, file, function
Tables	row groups	columns, units, source
Tickets	one issue thread	status, owner, date
Chats	turn windows	speaker, timestamp

Window pattern

Add overlap to preserve context and reduce boundary loss.

Chunking rules

- Keep source IDs.
- Avoid orphan headings.
- Respect access control.
- Rebuild on document changes.

Best quick test

Read the retrieved chunks without the answer.
If a human cannot answer from them, your model cannot reliably answer either.

Save this handbook for your 2026 GenAI builds

das-purnendu

EMBEDDINGS AND SEARCH

Search recipe

dense
semantic

+

sparse
keywords

+

filters
metadata

→

top_k
candidates

Store

- [x] chunk text
- [x] embedding
- [x] source id
- [x] created_at
- [x] ACL fields

Avoid

- [x] embedding only
- [x] missing source
- [x] stale docs
- [x] one index for all tenants

retrieval.py

```
results = search(query_vector, filter={"tenant_id": tid})
results = rerank(query, results)
context = pack(results, max_tokens=budget.context)
```

Save this handbook for your 2026 GenAI builds →

das-purnendu

RERANK + PACK THE CONTEXT

</> packing

```python
def pack(chunks, max_tokens):
    chosen, used = [], 0
    for c in dedupe(rerank(chunks)):
        n = count_tokens(c.text)
        if used + n > max_tokens:
            continue
        chosen.append(c)
        used += n
    return chosen

```

Pack order

- highest relevance
- source diversity
- freshness boost
- citation quality
- fits token budget

Citation contract

Answer sentences should map back to chunk IDs.

```
[chunk: policy-2026-p12] -> citation in final answer
```

Do not stuff context blindly

More context can lower answer quality. Rerank, dedupe, compress, and cite only what earns its place.

Save this handbook for your 2026 GenAI builds

das-purnendu

EVALS BEFORE DEPLOY

Layer	Measure	Python artifact
Retrieval	hit rate, MRR, citation recall	`golden_queries.jsonl`
Answer	schema, faithfulness, style	`pytest + judge rubric`
Safety	refusal, injection resistance	`red_team_cases.jsonl`
Ops	latency, cost, error rate	`trace assertions`

test_eval.py

```python
def test_answer_has_citations(golden_case):
    answer = app.answer(golden_case.question)
    assert answer.citations
    assert answer.confidence >= 0.70
    assert not contains_secret(answer.summary)
```

Best eval set

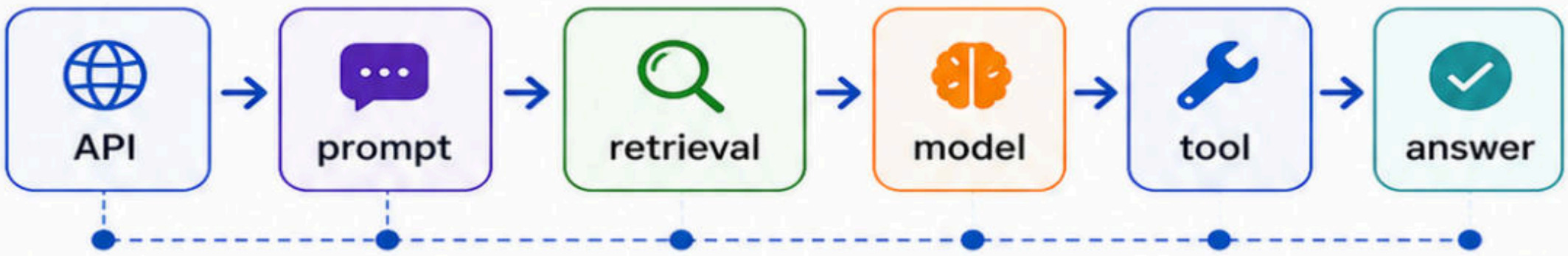
50 real user questions, 20 edge cases,
10 adversarial cases.
Small, sharp, and run on every serious prompt change.

Save this handbook for your 2026 GenAI builds

das-purnendu

OBSERVABILITY FOR AI SYSTEMS

Trace every request



Log

- [x] trace_id
- [x] user tier
- [x] model
- [x] prompt_id
- [x] token usage

Redact

- [x] API keys
- [x] PII
- [x] full documents
- [x] tool credentials

tracing

```
with tracer.span("llm.call", prompt_id=pid) as span:
    result = await model.chat(messages)
    span.set_metric("tokens", result.usage.total)
```

If you cannot **trace** it, you cannot **debug** it.

Save this handbook for your 2026 GenAI builds

das-purnendu

STREAMING UX WITH ASYNC GEN

stream.py

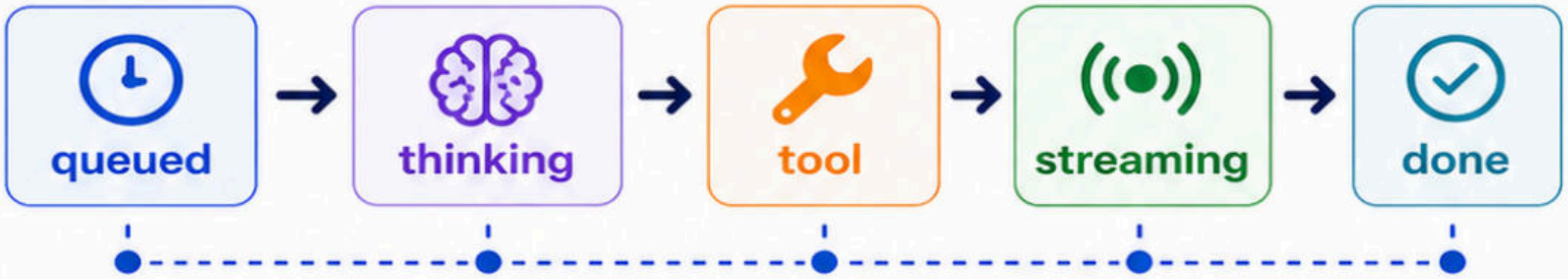
```python
async def stream_answer(question):
    async for event in model.stream(question):
        if event.type == "token":
            yield event.text
        elif event.type == "tool_call":
            yield render_status(event.name)


# SSE / WebSocket layer consumes it
```

UX rules

- first token fast
- show tool status
- handle cancel
- persist final answer
- retry from state

Lifecycle



Streaming is a product feature, not just an API option.

Save this handbook for your 2026 GenAI builds

das-purnendu

CACHING + MEMORY THAT STAYS HONEST

Cache key

```
hash(model, prompt_id, schema_version, inputs, tools)
```

Invalidate when any behavior-shaping input changes.

Response cache

- [x] same prompt + same inputs
- [x] short TTL for user data

Embedding cache

- [x] hash raw text
- [x] include model name

Conversation memory

- [x] summaries
- [x] facts with sources

Memory trap

Do not treat summaries as truth. Store claims with source, timestamp, confidence, and a way to forget them.

Cache speed. | Keep correctness. | Respect privacy. | Cite sources.

Save this handbook for your 2026 GenAI builds

20/24

das-purnendu

SECURITY FOR GENAI PYTHON

Secrets

- never in prompts
- load from vault/env
- redact logs

Prompt injection

- context is untrusted
- tool allowlist
- quote sources

File uploads

- scan type/size
- sandbox parsing
- strip active content

Tools

- least privilege
- human approval
- audit side effects

</> policy

```python
if tool.requires_approval and not user_approved:
    return NeedApproval(tool=tool.name, args=safe_args)

assert request.tenant_id == resource.tenant_id
```

Security is a product feature.
Design it before the first incident.

Save this handbook for your 2026 GenAI builds

das-purnendu

AGENTS DONE SANELY

Agent = loop + state + tools + stop conditions

plan → act → observe → reflect

Bound it

- [x] max steps
- [x] max cost
- [x] timeouts
- [x] tool quotas

Make state explicit

- [x] goal
- [x] scratchpad
- [x] facts
- [x] tool results

Review risky actions

- [x] payments
- [x] emails
- [x] deletes
- [x] external posts

Best first agent

A narrow workflow with 3-5 tools, a clear success condition, and an eval set. Expand only after it behaves.

Save this handbook for your 2026 GenAI builds

22/24

das-purnendu

TESTING AI PYTHON CODE

mocking

```python
class FakeLLM:
    async def chat(self, messages):
        return Answer(
            summary="ok",
            confidence=0.9,
            citations=["doc-1"]
        )

app = App(llm=FakeLLM())
```

Test pyramid

- pure functions
- schema validation
- retrieval fixtures
- model replay tests
- end-to-end smoke

Thing	Fake it with	Assert
LLM	FakeLLM / replay JSON	schema + behavior
Vector DB	small in-memory index	top_k and metadata
HTTP	mock transport	timeout and retries
Tools	dry-run adapter	policy and args

Mock the model. Test your Python.
Replay real failures.

Save this handbook for your 2026 GenAI builds

23/24

das-purnendu

PERFORMANCE PLAYBOOK

Bottleneck	Try first	Python move
LLM latency	stream + smaller context	async generator
Embedding jobs	batch + limit concurrency	Semaphore
Retrieval	metadata filters + HNSW tune	measure top_k
JSON parsing	validate once	Pydantic boundary
CPU parsing	parallel workers	ProcessPool

Python 3.14 note

Free-threaded Python is officially supported in 3.14, but production gains depend on your dependencies. Benchmark your exact workload.

measure first

```python
with timer("retrieve"):
    chunks = await retriever.search(query)
    metrics.histogram("rag.retrieve_ms", timer.ms)
```

Measure before optimizing.
Optimize the slowest real path.

Save this handbook for your 2026 GenAI builds

24/24

das-purnendu

PRODUCTION CHECKLIST

Before merge

- [x] Typed request and response schemas
- [x] Timeouts, retries, and rate limits
- [x] Token and cost budgets per feature
- [x] Prompt versions with eval cases
- [x] Retrieval metrics and citation checks
- [x] Structured output validation

Merge only code you would be confident to debug at 3am.

Before launch

- [x] Tool allowlist and approval policy
- [x] Secret redaction in logs and traces
- [x] Streaming cancellation path
- [x] Replay tests for model failures
- [x] Dashboard for latency, cost, errors
- [x] Rollback plan for prompt/model changes

Ship features. Ship guardrails. Both or neither.

30-day practice plan

Week 1	build a typed chat API
Week 2	add RAG with citations
Week 3	add evals, tracing, and caching
Week 4	harden tools, security, and deployment

You do not need more magic.
You need reliable Python around the magic.

Save this handbook for your 2026 GenAI builds