### **✅ 1. Write Unit Tests for Input Assumptions**

Your function create\_faiss\_index\_from\_services() **assumes each item is a dict** with a specific structure. That should be tested.

#### **✅ Test: "Valid Chunk Format"**

python

CopyEdit

def test\_valid\_chunk\_structure():

valid\_chunk = {

"embedding": [0.1, 0.2, 0.3],

"text": "Some service description",

"metadata": {"location": "St Albans"}

}

try:

create\_faiss\_index\_from\_services([valid\_chunk])

except Exception as e:

assert False, f"Valid chunk caused error: {e}"

#### **✅ Test: "Reject Strings Instead of Dicts"**

python

CopyEdit

def test\_invalid\_chunk\_string():

with pytest.raises(TypeError):

create\_faiss\_index\_from\_services(["this is not a dict"])

### **✅ 2. Add Error Handling in the Function**

Wrap chunk access in validation logic. Example:

python

CopyEdit

def create\_faiss\_index\_from\_services(chunks):

all\_embeddings = []

id\_to\_metadata = {}

for i, chunk in enumerate(chunks):

if not isinstance(chunk, dict):

raise TypeError(f"Chunk at index {i} is not a dictionary: {chunk}")

if 'embedding' not in chunk:

raise ValueError(f"Missing 'embedding' field in chunk at index {i}")

if not isinstance(chunk['embedding'], list):

raise TypeError(f"'embedding' must be a list at index {i}")

all\_embeddings.append(chunk['embedding'])

id\_to\_metadata[i] = chunk.get("metadata", {})

# FAISS logic continues...

### **✅ 3. Validation After Loading JSON**

As a safety net during deserialization:

python

CopyEdit

chunks = []

with open("service\_chunks.jsonl", "r") as f:

for i, line in enumerate(f):

try:

chunk = json.loads(line)

if not isinstance(chunk, dict):

raise ValueError(f"Line {i} is not a dict")

chunks.append(chunk)

except json.JSONDecodeError as e:

print(f"Error decoding line {i}: {e}")

### **✅ 4. Consider Writing a Schema Validator**

For more complex use cases, use pydantic or cerberus to define a schema and validate chunks before using them.

### **✅ Summary: What This Tells You**

* Functions should **never assume structure blindly** – validate or test it.
* Unit tests should cover **expected and malformed inputs**.
* Reading/parsing code should always have **robust error handling** with helpful debug messages.
* As your project scales, **input validation becomes more critical** — this problem is a helpful early warning to put it in place now.

Let me know if you'd like help writing a proper validate\_chunk() helper or a pydantic model for your data.