hypergraph

How to Elegantly Build Complex AI Workflows

DS Leads Meetup, January 2026

Gilad Rubin

Introduction

- Independent AI Engineer
- 12+ years experience in Data Science & AI
- Obsessed over how to build elegant ML & AI systems


JUPYTER
NOTEBOOKS
PYTHON
What the hell is this?

We've all been there...

Slow

- **Run this script** → then that cell → then the next notebook
- Copy-paste configs into files
- Changed a config? **Rerun everything** from the start

Brittle

```
# config.yaml
model_name: "gpt-5.2"

# train.py
LEARNING_RATE = 0.001

# .env
OPENAI_MODEL=gpt-5.2-mini

# notebook cell 47
params = {"lr": 0.01, "epochs": 50}
```

AI Slop

```
# AI-generated code that "looks right"

def process_data(content, model="claude-3-5-sonnet"):  # ← outdated model
    chunks = content[:200]          # ← why 200? magic number
    result = llm.generate(chunks)
    return result
```

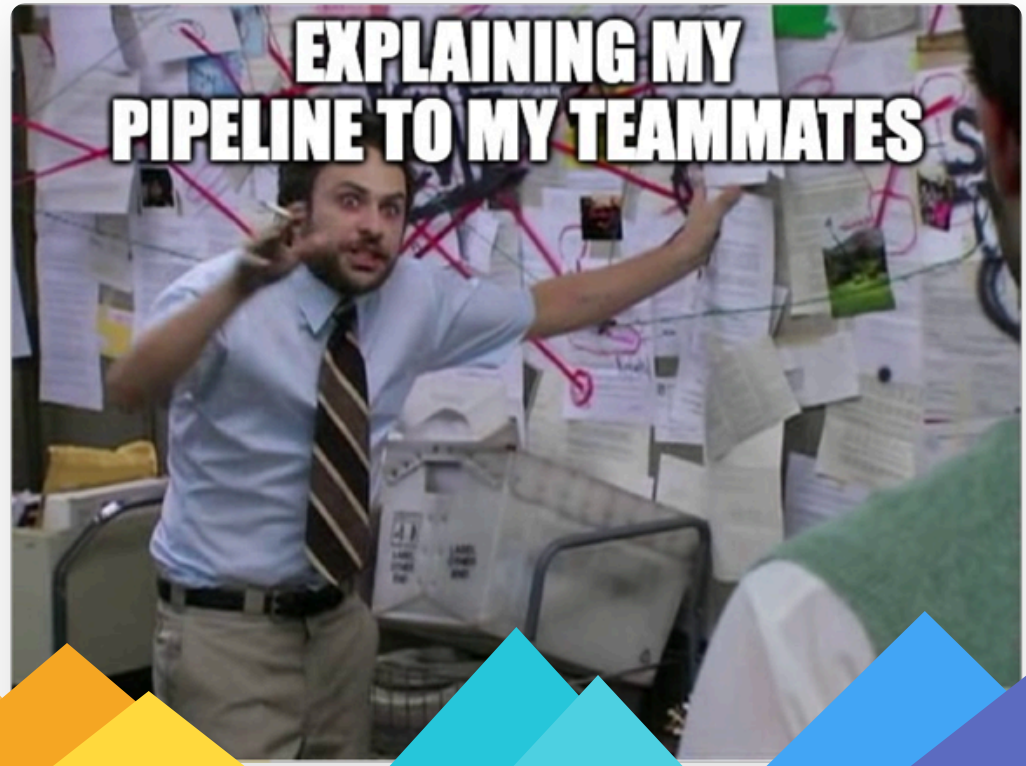
Outdated models, magic numbers, no explanation — but it runs.

AI Slop

```
class ChunkingService:  # ← duplicates existing functionality
    def __init__(self, max_tokens=200):
        self.max_tokens = max_tokens
    def chunk(self, content):
        return [content[i:i+self.max_tokens]
                for i in range(0, len(content), self.max_tokens)]
```

Re-implements things that already exist in your codebase.

Collaboration & Handoffs



Maintainability

- **No clear entry point** — which file/script do you even run first?
- **Implicit dependencies** — "run this before that" lives in someone's head
- **Unreadable diffs** — notebooks: base64 noise; scripts: spaghetti changes
- **Testing?** — "I ran it and it looked right"

Production Readiness

- **Hardcoded paths** — `~/Desktop/data_final_v3.csv`
- **Missing dependencies** — "it works on my machine"
- **No error handling** — fails silently or crashes spectacularly
- **Can't scale** — works for 10 items, breaks at 10,000

Root Causes

- No Methodology
- No Structure
- No Tools

There is a way out

The Journey Out of this Mess

Two Leaps:

- From Notebooks & Ad-Hoc Scripts → Explicit Graphs
- From Explicit Graphs → HyperGraph

Plus:

- **Methodology** — Code Structure & Design Patterns

Leap 1: Explicit Graphs

Our Work is Natively a Graph

Take input → do something → produce output → feed to next step

That's a graph.

But We Write It as Regular Code

```
docs = retriever.search(query)
response = llm.generate(query, docs)
```

We're taking a graph-shaped problem and expressing it with arbitrary code.

What We're Missing

- Can't visualize the flow
- Can't cache intermediate results
- Can't parallelize independent steps
- Can't isolate errors to specific nodes

Express Your Code as a Graph

```
retrieve = Node(func=retriever.search, name="retrieve")
generate = Node(func=llm.generate, name="generate")

edges = [Edge(from_node="retrieve", to_node="generate")]

graph = Graph(nodes=[retrieve, generate], edges=edges)
graph.run(query=query)
```

*Same logic, but now Python **KNOWS** it's a graph.*

The Gifts of Explicit Graphs

- **Caching** — ran retrieve once, skipped it the second time
- **Logging** — each node reported inputs/outputs automatically
- **Parallel execution** — independent nodes run concurrently
- **Error isolation** — know exactly which node failed

The Code

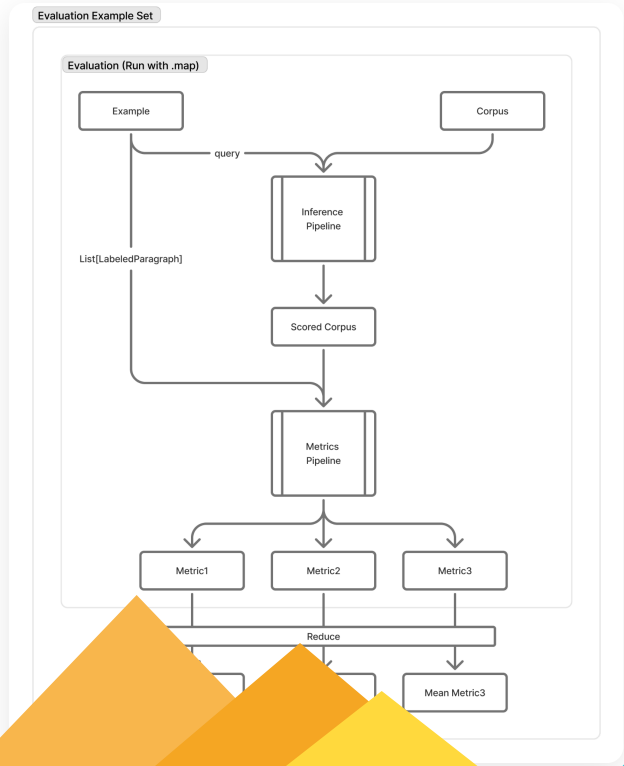
```python
@node(output_name="query_embedding")
def embed_query(query: str, embedder: Embedder) -> list[float]:
    return embedder.embed(query)

@node(output_name="candidate_docs")
def vector_search(
    query_embedding: list[float], vector_store: VectorStore, top_k: int
) -> list[dict]:
    return vector_store.search(query_embedding, top_k=top_k)

@node(output_name="documents")
def rerank(candidate_docs: list[dict], min_score: float) -> list[str]:
    return [d["text"] for d in candidate_docs if d["score"] >= min_score]
```

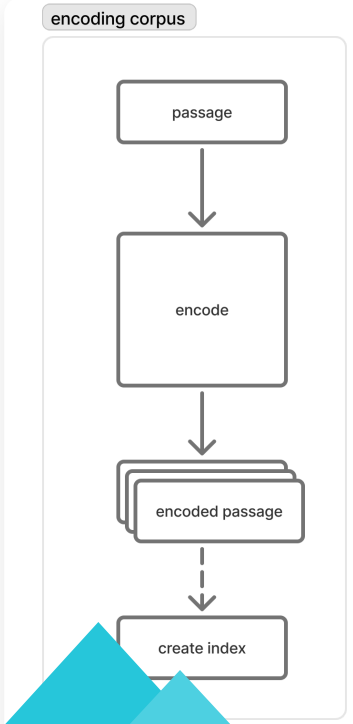

embedder
query
embed_query
→ query_embedding
top_k
vector_store
vector_search
→ candidate_docs
min_score
rerank
→ documents

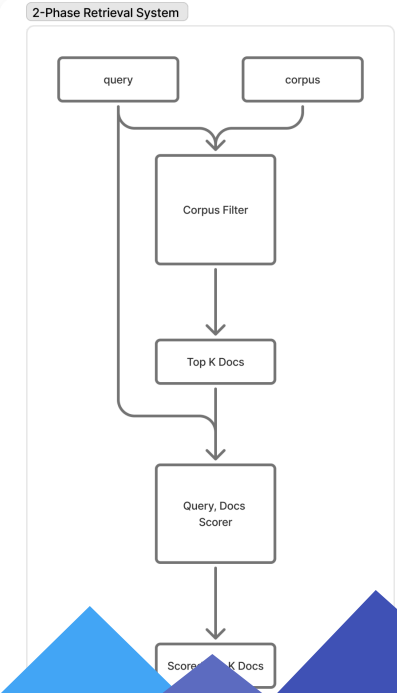
Comprehension

- **Your own work** — come back 3 months later, the graph is still clear
- **Your teammates' work** — onboarding in minutes, not days
- **Stakeholders** — show them the graph, not the code

Drawing --> Code







Code --> Drawing

AI modifies code → graph updates automatically → you verify the graph, not the code

Try Explicit Graphs

Moving to graphs alone solves many problems:

DAG Frameworks:

- Kedro
- Hamilton
- Pipefunc
- Prefect

Interactive:

- LangGraph
- AutoGen
- Burr, Strands...

But I kept hitting walls...

Three Things I Couldn't Get

- **Nested graphs as first-class citizens** — graphs that contain graphs
- **Elegant scaling**
- **DAGs + Cycles** — same framework for pipelines AND agents

hypergraph

A Unified Framework For
Hierarchical & Modular Workflow Orchestration

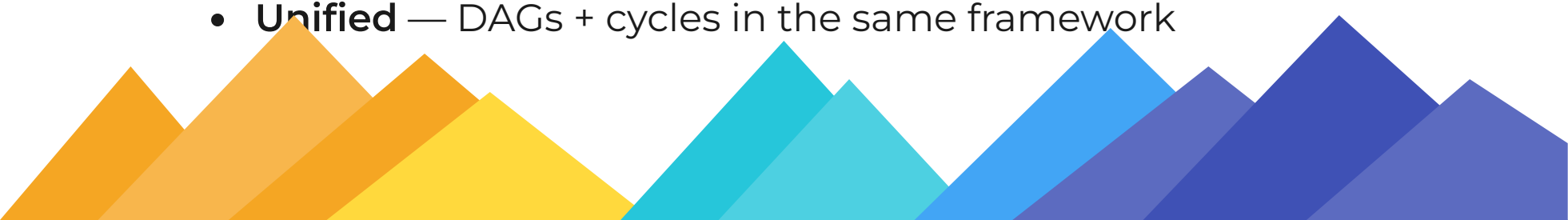
What is HyperGraph?

A Python Framework for AI/ML Workflows

- Open-source

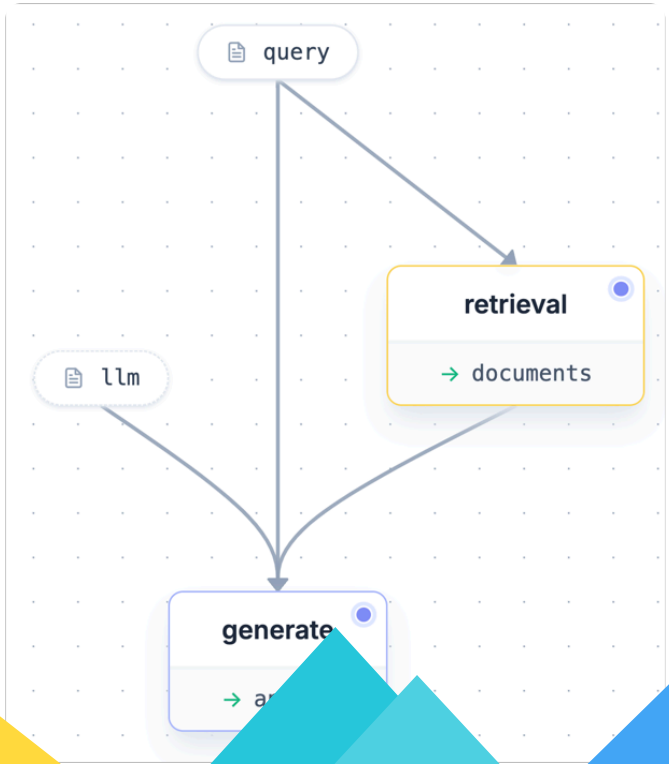
Three innovations:

- **Hierarchy** — nested graphs as first-class citizens
- **Scale with** `.map()`
- **Unified** — DAGs + cycles in the same framework

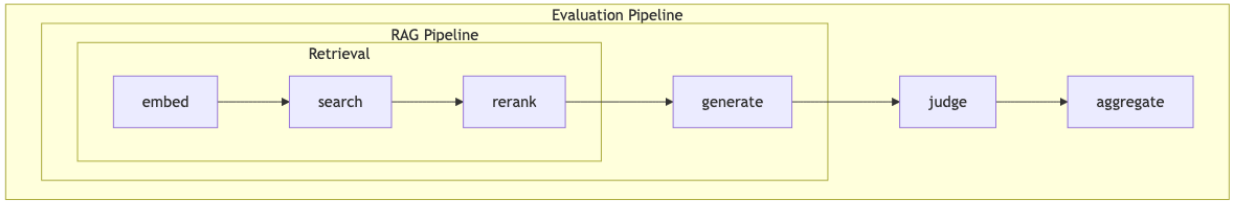


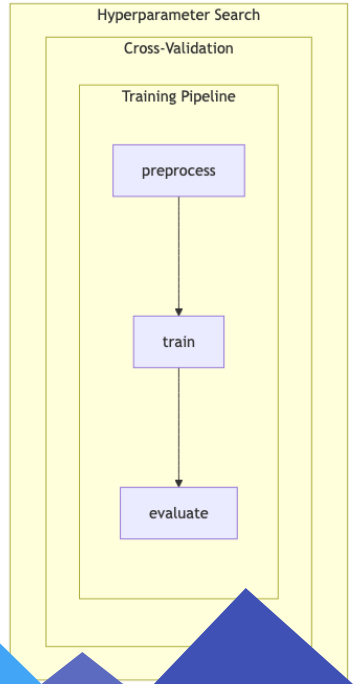
What Does This Graph Do?

And Now?



Our Work is Hierarchical!





1. Hierarchy | **2. Scale with .map()** | 3. Unified

What about scale?

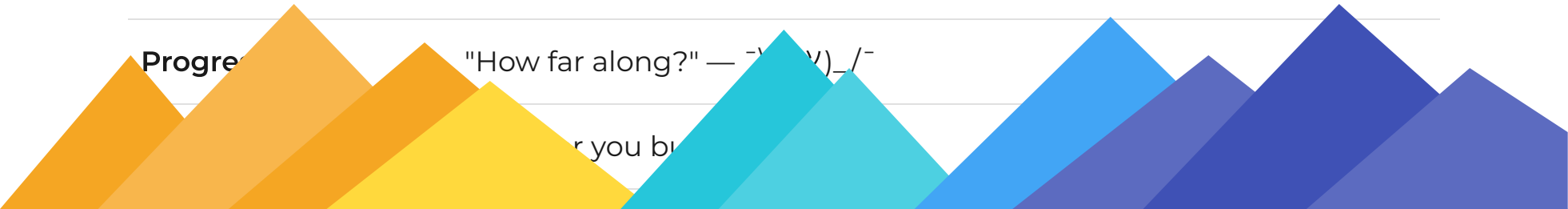
The Batch Rewrite Problem

```python
# You wrote this (clean):
def retrieve(query: str) -> list[Document]

# Now you need this (nightmare):
def retrieve(queries: list[str]) -> list[list[Document]]

# And this cascades:
def pipeline(queries: list[str]) -> list[list[list[ScoredDocument]]]
```

Rewrite every function. Batch logic everywhere.

What You Lose with Manual Batching

Problem	Pain
Testing	Can't unit test one item cleanly
Errors	"Item 847 failed" — which input? what state?
Debugging	Re-run entire batch to reproduce
Performance	You manage parallelism, retries, rate limits
Progre	"How far along?" — ¯_(ツ)_/¯
	r you bu



Scale with `.map()`

```
# .run() — single item
result = await runner.run(rag_graph, {"query": "What is HyperGraph?"})

# .map() — same graph, many items. That's it.
queries = ["What is HyperGraph?", "How does .map() work?", ...]
results = await runner.map(rag_graph, {"query": queries}, map_over="query")
# Automatically: parallelized, cached, error-isolated, with progress!
```

The only change: `.run()` → `.map()`. No code rewrite. No batch logic.

DAGs Can't Loop

"I built a beautiful RAG pipeline. Then PM says: we need multi-turn conversations."

DAGs + Cycles — Unified

```python
# My RAG pipeline (DAG) — unchanged
rag_graph = Graph([embed_query, retrieve, generate], name="rag")

# Route: loop back or end
@route(targets=["rag", END])
def should_continue(query: str) -> str:
    return END if query.strip().lower() == "exit" else "rag"

# Wrap it with conversation logic — add a cycle
chat_graph = Graph(
    [rag_graph.as_node(name="rag"), accumulate_history, ask_user, should_continue],
    name="rag_chat",
)
```

The existing RAG graph didn't change. Same patterns. Same testing appro

Op ll spectrum: da ines AND a

Live Demo

- **Notebook 01** — Basic RAG pipeline (build nodes, assemble graph, visualize, run)
- **Notebook 02** — Nested graphs / hierarchy (compose sub-graphs, drill-down viz)
- **Notebook 03** — Batch with `.map()` (single-item → collection, parallel execution)
- **Notebook 04** — Cycles & InterruptNode (v… in…
  …p)

The Methodology

Codebase Structure

```
src/project/
├── models.py          # Data contracts (business language)
├── prompts/           # Prompt templates (optional)
│
├── components/        # Reusable building blocks
│   ├── llm.py         #   LLM wrappers (swappable)
│   ├── embedder.py    #   Embedding models
│   └── vector_db/     #   Vector store integrations
│
├── graphs/            # HyperGraph definitions
│   ├── retrieval.py   #   Retrieval sub-graph
│   ├── generation.py  #   Generation sub-graph
│   ├── rag.py         #   Full RAG (composes above)
│   └── conversation.py #   Multi-turn agent
│
└── cli/               # CLI runner
```

Components

```python
class Embedder:
    def __init__(self, model, dimensions, ...): ...

    async def embed_text(self, text: str) -> list[float]: ...

class LLM:
    def __init__(self, model, temperature, ...): ...

    async def generate(self, messages: list[dict]) -> str: ...

# Hypster config in the same file — defines hyperparameters
def embedder_config(hp: HP) -> Embedder:
    model = hp.select(["text-embedding-3-small", "text-embedding-3-large"], ...)
    dimensions = hp.select([1536, 3072], default=1536, name="dimensions")
    return Embedder(model=model, dimensions=dimensions)
```

Models

```python
@dataclass
class DocumentPage:
    document: DocumentInfo
    page_number: int
    content: str
    embedding: list[float] = field(default_factory=list)

@dataclass
class RetrievedDocument:
    doc_id: str
    title: str
    pages: list[DocumentPage]
    page_scores: dict[int, float] = field(default_factory=dict)

@dataclass
class RAGResponse:
    query: str
    status: Lit cess", "re ected", "requires_visual"]
    answe lo
    ou nt | None = None
```

Graphs

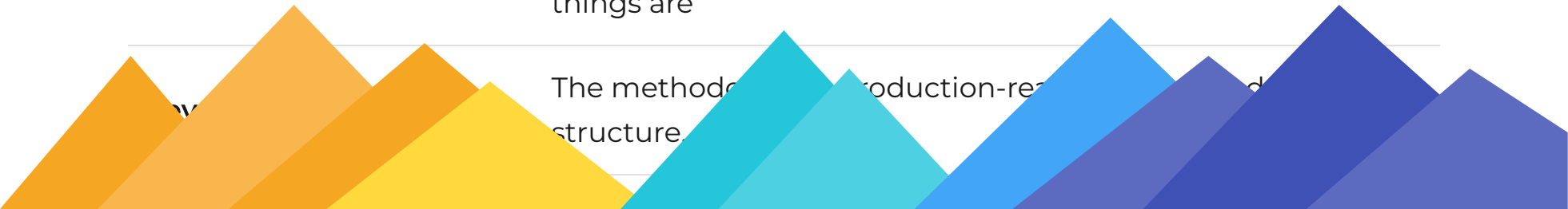
```
# graphs/retrieval.py — works standalone
retrieval_graph = Graph(
    [retrieve_pages, aggregate_pages, sort_documents, limit_documents],
    name="retrieval",
)

# graphs/rag.py — composes sub-graphs
rag_graph = Graph([
    retrieval_graph.as_node(name="retrieval"),
    generation_graph.as_node(name="generation"),
], name="rag")
```

Nodes

```python
@node(output_name="retrieved_pages")
async def retrieve_pages(
    query: str, vector_store: VectorStore, top_k: int
) -> list[ScoredPage]:
    return await vector_store.search(query, top_k=top_k)

@node(output_name="documents")
def aggregate_pages(
    retrieved_pages: list[ScoredPage], aggregator: Aggregator
) -> list[RetrievedDocument]:
    return aggregator.aggregate(retrieved_pages)
```

Full Circle

Problem	How It's Solved
Understanding your own code	Visualization — the graph IS the documentation
Working with AI agents	Shared vocabulary — draw → generate → verify
Fragile experimentation	Hypster configs — safe, explicit, introspectable
Collaboration & handoff	Same structure everywhere — new person knows where things are
...	The method... ...oduction-re... ...d structure.



Caveats & What's Next

- HyperGraph is in **early alpha** — things are moving fast
- It is being used in production in hospitals in Israel
- If it's giving you value, experiment with it — and talk to me
- Working with explicit graphs is a whole topic — today was the introduction

Looking Forward

- Improving the codebase and methodology
  - Would love to get help from open-source maintainers
- Much more features & tighter integrations
- Blogs, articles, videos, tutorials

Questions?

Thank you!

me@giladrubin.com

github.com/gilad-rubin/hypergraph

github.com/gilad-rubin/hypster