

Flanks lambdaloopers

The GenAI Playbook

Building GenAI for wealth data

How to turn complex, fragmented portfolios into a secure, traceable and genuinely useful conversational experience. The technical and strategic calls behind it.

This Playbook summarises the technical and strategic decisions shared by Héctor Borobia, Head of AI at Flanks, in the **IA lo hicieron** interview series by LambdaLoopers.

01 From manual reporting to financial chat-to-data

Flanks built the **wealth-data infrastructure** behind a new generation of AI. Its modular platform automates the aggregation and enrichment of complex, fragmented portfolios and turns them into a **single, high-quality data layer**, a genuine 360° view across every asset class. The mission is to *democratise wealth management:* strip out manual friction so family offices, private banks, asset managers and fintechs can focus on faster, smarter, more personalised advice.

Before

- Manual portfolio queries
- Excel exports and ad-hoc reports
- Reliance on manual technical analysis
- High friction to cross portfolios, currencies and benchmarks

After

- Natural-language questions about portfolios
- Real-time currency-exposure analysis
- Automatic comparison against benchmarks like the S&P 500
- Near-real-time report preparation

Key insight

GenAI strips away the friction of reaching value Flanks already held, high-quality financial data.

02 Productive GenAI isn't just AI, it's product, data, software and domain

The hard part was never calling a model over an API. It was working out **which data the advisor actually needs**, where the risk sits, and how to hand back an answer they can trust. Five disciplines have to work as one.

Functional discovery

Define precisely which questions the system should answer, and which it shouldn't.

Data design

Know what data exists, where it lives and at what quality before modelling.

Software engineering

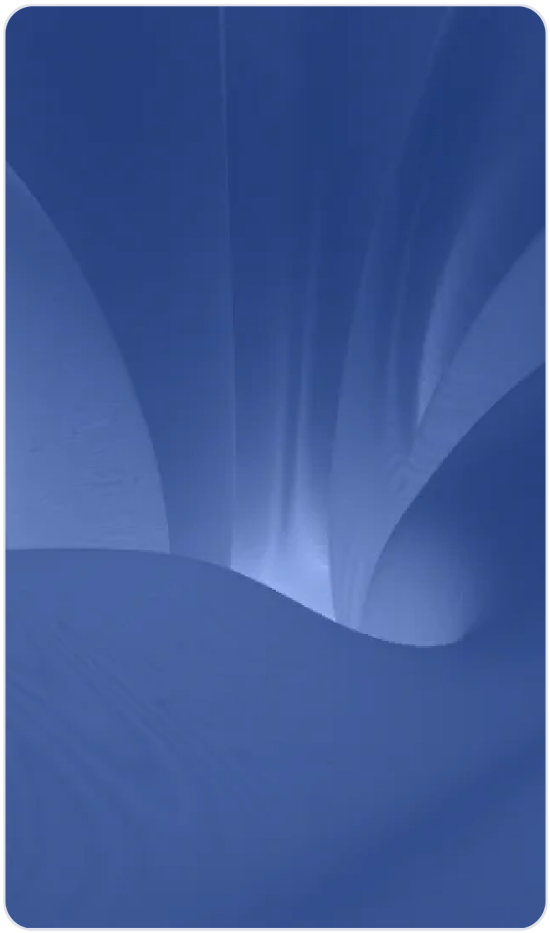
Integrate APIs, tools, traces and existing systems robustly.

Financial domain

Interpret portfolios, benchmarks, risk and reporting with judgment.

Conversational UX

Handle ambiguity and clarifications, and build confidence in the answer.



Key decision Treat GenAI as a **product and data project**, not a one-off AI experiment.

03 The model matters, but less than the data system around it

Choosing a model is the easy decision. The durable advantage is in what you wrap around it: the context you feed, the tools you wire in, and how gracefully you handle ambiguity.

Model selection

Start with a large model to isolate variables. Don't choose on generic benchmarks, measure quality, latency, cost and correct tool use, and use different models for different tasks.

Context engineering

Prompting decides **what context enters, in what order**, and which tools are wired in. More context isn't always better, it adds noise, cost and errors.

Frameworks

LangGraph, LlamaIndex, CrewAI and agent SDKs take care of boilerplate and state graphs. What they *won't* do is design your product architecture for you.

Conversational UX

When a question is ambiguous, the system should ask, not guess. If a stock trades on several exchanges, the right move is: *"Which market do you mean?"*

Key decision Don't chase *"the best model."* Pick the **right model for the right task**, inside an architecture you can measure.

04 Tools are the frontier between probabilistic AI and deterministic business

A production-grade tool has to be built so the model **knows it exists, knows when to reach for it**, gets clean arguments, and can recover gracefully when something breaks.

Semantic design

Easy-to-interpret inputs, structured outputs, clear function descriptions, and permissions defined per user and data domain.

Functional errors

No data for a given year? The tool doesn't throw *"error"*, it replies *"No values for that period; try another range,"* so the model can simply retry.

Deterministic architecture

Critical logic lives in **controlled tools**. The LLM interprets and writes; the calculations, permissions and real data come from deterministic systems.

Prefiltering & validation

Programmatic validations before the model. Field prefiltering to avoid passing unnecessary noise. Latency measured per tool.

Key insight

A badly designed tool makes the whole agent look clumsy. A **well-designed one quietly removes most of the system's risk.**

05 One protocol to connect any model to your data and tools

A good MCP should work across several models. If it only works with the most powerful one, it's probably mis-specified.

On treating MCP as an infrastructure product

+ What MCP brings

- Standard connection between models, tools and data
- Reusable financial capabilities, exposed
- Clients connect their own agents to Flanks data
- Separation of the end interface from the pluggable data layer
- A potential new business model: the *"data engine"*

Mandatory controls

- OAuth + access tokens + client IDs
- Allow lists and auditing of tool calls
- Usage limits and latency observability
- More MCPs = more context, more cost, more noise

Key decision Treat MCP as an **infrastructure product**, not just a technical connector.

06 Memory isn't storing more, it's retrieving the right context

01 · Capture

Working memory

The live conversation, the context of the session thread in progress.

02 · Distil

Summary on close

An LLM captures the key points and decides which candidates are worth keeping.

03 · Store

Long-term memory

Milestones and advisor patterns: benchmarks, favourite formats, risk profile.

04 · Retrieve

Contextual retrieval

A tool pulls the useful context back in exactly when the user needs it.

"The AI learns as you use it" is tempting, and misleading. A frozen model doesn't learn on its own; the experience gets better because the system **retrieves and injects the right context** at the right moment, with no retraining at all.

Key decision

Reach for **context as memory** before you reach for retraining, unless the ROI clearly makes the case.

07 Not everything needs agents, sometimes the best AI is a workflow

Orchestration starts with one blunt question: *is the task deterministic?* If the inputs and the steps never change, you almost certainly don't need a multi-agent system. Let complexity earn its place.

Workflow

Fixed sequence

Predictable inputs and steps. No real-time decision needed.

Single agent

Real-time decision

Little specialisation. ReAct or simple patterns usually suffice.

Multi-agent

Specialised subtasks

Only for parallelisable, aggregable subtasks with real specialisation. More agents = more error surface.

Modular design

Flanks works with acyclic graphs and decoupled components, adding tools or capabilities without redesigning the whole system.

Traditional ML as an ally

Not everything should be solved with LLMs. BERT-style classifiers can be better for repetitive, fast, deterministic tasks.

Key decision Let the **task** decide between workflow, single agent or multi-agent, not the hype.

08 Real production means traces, evals, guardrails and controlled cost

AgentOps is what turns a slick demo into a system you can actually run. Observability belongs in the POC, **not bolted on at the end.**



Tracing

Instrument input, tool calls, output, tokens, latency per step, errors, retries, loops and timeouts.

Evals & golden set

Create 15 to 50 questions with expected answers from the start. Use LLM-as-a-Judge with a different model to evaluate at scale.

Guardrails

Input: block out-of-domain questions. Output: block unsafe answers. Behaviour: limit loops and unwanted actions.

Feedback & FinOps

User feedback feeds prompt and golden-set improvements, it doesn't train the model. Measure tokens, calls and cost per client.

Key insight No traces, **no debugging.** No evals, **no improvement.** No guardrails, **no operational control.**

Seven decisions for taking GenAI to production in wealth-tech

Flanks' core lesson: shipping GenAI isn't about plugging in an LLM. It's about **designing the system around it.**

01 **Validate the problem first**

Start from a real pain: reducing the friction of accessing complex wealth data.

02 **Start small so you can measure**

One model, one tool, few MCPs, a scoped case. If all fails at once, you can't tell what broke.

03 **Separate generation from logic**

The LLM interprets and drafts. Tools calculate, validate, query and apply permissions.

04 **Design tools and MCPs as product**

Exposing APIs isn't enough. Design them so models understand them without breaking security.

05 **Use context as a competitive edge**

The model gets commoditised. The differentiator is data, context, memory and permissions.

06 **Don't over-engineer with agents**

If a workflow solves it, use a workflow. Multi-agent only for real specialisation.

07 **Instrument from the POC**

Tracing, golden set, evals, feedback, guardrails and costs must exist before scaling.

The model isn't the moat. The moat is the data, the tools, the orchestration, and the ability to run GenAI under control.



Flanks

Wealth, easily managed

Let's talk!

Flanks.io