Here’s a professional pitch-style reply you could send back to someone who asked you to “replicate Full Reval calculation in a Python tool called SVaR Factor Attribution Model (SFAM).”

**Pitch Reply**

Replicating **Full Reval** inside SFAM is possible, but we need to be clear on scope and complexity.

* **What Full Reval actually does**: it’s not just a P&L formula—it’s a *workflow*:
  1. Load full market data (curves, surfaces, spreads).
  2. Apply historical or stress **scenario bumps** to every risk factor simultaneously.
  3. Re-price each trade with the bumped data using the **original front-office pricing models** (Calypso, GMI, Broadridge, etc.).
  4. Compute stressed P&L as the difference between bumped and base NPVs.
* **Why this matters**: unlike delta-gamma approximations, Full Reval captures true **non-linearities**, **path dependencies**, and model-specific nuances. That’s why it drives **SVaR capital numbers**.
* **Challenges in Python**:
  1. Every product family has a different pricer with unique inputs (swaps, options, futures, credit, structured).
  2. Market-data objects aren’t uniform; curves, vols, credit spreads, and FX surfaces all need consistent bumping and re-building.
  3. You’d need to create a wrapper/API layer for each pricer or re-implement pricer logic directly—essentially rebuilding Vasara/EMR plumbing.
* **Where SFAM fits**:
  1. SFAM is well-suited for **factor attribution, scenario scaling, and what-if analysis**.
  2. It can consume **delta-gamma style outputs** and let you explore P&L drivers at factor level.
  3. But replicating *true* Full Reval inside SFAM means embedding or interfacing with the pricing engines—an order of magnitude more complex than attribution.

**Pitch framing**:  
👉 My recommendation is to use SFAM for **factor-based insight and optimization** (e.g., “what if I scale Greeks, scale notional, replace hedge with ETF”), while leaving **Full Reval generation** to the existing pricing stack. Then SFAM can consume those Full Reval P&Ls and break them down across factors.

That way, SFAM becomes **value-add**—fast, transparent, exploratory—without trying to rebuild an entire risk engine in Python.

Would you like me to also draft a **shorter “executive pitch” version** (2–3 sentences) that you can use in a meeting without all the technical detail?

Here’s a clean, structured rewrite of your “Full Reval demo” with all the points and nuances called out.

**Full Revaluation (Full Reval) — Demo Script (Rewritten)**

**1) What this template does**

* Loads market data, applies scenario **bumps**, and prices a variety of trades for **full revaluation** (stress and VaR-style runs).
* It isn’t a universal, one-size-fits-all workbook; **pricing tabs are often set up ad-hoc** per position due to model/input variability.

**2) Key tabs and roles**

* **Configuration** (primary control panel)
  + Set **valuation date**.
  + Choose **market data sets** to load (curves, vol surfaces).
  + Configure **curve-bump settings** (scenario type, sources).
  + Triggers market-data load + bump application (when GXL is up).
* **Full Jacobian / Per-bump MD tabs**
  + Where the **bump definitions** are enumerated and applied.
* **Pricing tabs (per product/position)**
  + Two essential cells per tab:
    - **Base NPV** (unbumped).
    - **Stressed P&L** (via **bumped NPV** minus **base NPV**).

Note: Most control happens on **Configuration**; downstream tabs update accordingly.

**3) System dependencies & current limitation**

* Uses **GXL** to load **market data** and **trades** (Calypso/GMI/etc.).
* In the demo, **GXL was unstable**, so execution couldn’t be run—but the process remains the same.

**4) How pricing is wired**

* **Base NPV**
  + Typically priced via **Calypso** (for most rates/vanilla positions).
  + Inputs pulled from a **flip/so-log** (price-input logger) to ensure the Excel/Calypso setup matches production pricing.
  + Example inputs: curve handle(s), valuation date, discounting curve, forecasting curve(s), conventions.
* **Shifted (bumped) NPV**
  + Start from the **exact base formula**, then **replace all market-data references** with references to the **bumped market data** objects.
  + Bumped objects are collected on a dedicated **market-data tab** (e.g., “Curve 1 (bumped)”).
  + **Stressed P&L = NPV(bumped) − NPV(base)** (same as VaR P&L aggregation pattern).

**5) How bumps are generated and applied**

* **Scenario source**
  + **VaR**: 1-day historical shifts from **time series** (e.g., SOFR forward bucket t vs t-1).
  + **Stress**: as defined in **scenario tables** (can be absolute/relative, level/slope, custom shapes).
* **Mechanics**
  + A function call assembles the **list of bump specs** (bucket → shift).
  + **All buckets are bumped simultaneously** to produce **one bumped curve** (full reval).
  + Contrast with **Greeks/Jacobian**: bump **one bucket at a time** with small shifts (delta/gamma approximation).
* **Independence from trade setup**
  + Bumps depend on **market-data history/scenario definitions**, **not** on how a particular trade is configured.
* **Discount factors vs curves**
  + The template doesn’t grab discount factors directly; it **bumps the curve** and re-computes discount factors **using the bumped curve** at valuation.

**6) Full Reval workflow (end-to-end)**

1. **Set valuation date** on Configuration.
2. **Select** curves and surfaces to load (Calypso/GMI/etc.).
3. **Define/choose scenario** (VaR 1D, specific stress, etc.).
4. **Load market data** (GXL) and **apply bumps** (build bumped curves/surfaces).
5. On product pricing tab:
   * Compute **Base NPV** using base curves/surfaces.
   * Compute **Bumped NPV** by referencing the **bumped** curves/surfaces.
   * **Stressed P&L = Bumped NPV − Base NPV**.
6. **Aggregate** across trades for portfolio results (outside this template).

**7) Product/model coverage & template strategy**

* **Calypso** covers most **rates** positions (swaps, caps/floors, vanilla options, etc.).
* **GMI** covers **exchange-traded** products; a **stable, product-specific tab per product type** exists (e.g., SOFR futures/options, bond futures/options across currencies).
* **Broadridge** covers **government bonds/treasuries**.
* For **Calypso-priced** trades, a **single generic tab is impractical**:
  + Variants: LIBOR vs EURIBOR vs BMA vs SOFR; float-float, fixed-float, fixed-fixed; currency, day-count, compounding, convexity, collateral discounting, etc.
  + A “universal” formula would explode into a long chain of conditionals.
  + **Practice:** clone/modify a **similar tab** or create a **new tab** per test.

**8) Example (interest-rate swap)**

* **Inputs**: trade object, valuation date, discounting/forecast curves.
* **Base NPV**: price with **base curve**.
* **Bumped NPV**: identical pricing call, but curve argument swapped to the **bumped curve** created by scenario shifts.
* **P&L**: difference between the two NPVs.

**9) Why Full Reval vs Delta-Gamma**

* **Delta-Gamma** is a **Taylor approximation** using small bucket shifts and Greeks.
* **Full Reval**:
  + Applies the **actual scenario bumps** all at once across the curve/surface.
  + **Reprices** the position with **true model non-linearities** and **path dependencies** (where applicable).
  + Avoids approximation error at the cost of **higher compute + more plumbing**.

**10) EMR → MARS & capital implications**

* Since **EMR has folded into MARS** and feeds **capital**, teams may try **“what-if” hedges** (book options/swaps, scale Greeks or notionals).
* Because **Full Reval** drives **SVaR** (and capital), you need **deep understanding** of:
  + Which **models** (front-office pricers) are used per asset class.
  + How **risk factors** map to those models.
  + How **scenario bumps** transform **market objects**.
  + How **trade inputs** (conventions, calendars, compounding, collateral) affect NPV.

**11) Replicating Full Reval in Python — feasibility & complexity**

* **Possible**, but a **large build**:
  + You’d need **wrappers for each pricer** (or a uniform pricing API) and a robust **market-data abstraction** (curves, vols, credit surfaces, FX, etc.).
  + Must support **scenario construction**, **bucketed shifts**, **build bumped MD objects**, and **re-wire pricing inputs**.
  + Account for **system heterogeneity** (Calypso vs GMI vs Broadridge), **edge cases**, **conventions**, and **exotic model inputs**.
* Realistically, you’re **re-creating Vasara/EMR-like plumbing** around model invocations and market-data transforms.

**12) Debugging & developer hooks (Vasara/MARS)**

* There is a **central entry point** (e.g., a *run-context harness*):
  + Inputs: **valuation date**, **system of record** (Calypso, GMI, Broadridge, OPICS), **trade ID**, and **metric** to run (Base NPV / Delta / VaR / Stress).
  + It **discovers dependencies** (market data, reference data) and **dispatches** to the right pricers.
* **Debug mode** lets you **step through**:
  + Verify **loaded conventions**, calendars, trade legs, discount/forecast curve choices.
  + Inspect **market objects** pre-/post-bump.
  + Find **mismatches** between local and official (MARS/EMR) results.
* **Caveat**: Finding the **right breakpoints** requires **experience**; details live across multiple code paths.

**13) Coverage maturity & gaps**

* The harness supports the **vast majority** of positions and variations, but **not all** are fully correct or complete.
* Some product corners require **custom review** (e.g., specific futures, FRAs, mid-curve options, exotic swap features).

**14) Learning curve & SME expectations**

* Even for a strong engineer/analyst: expect **~4 months of focused work** to become effective at replicate/debug for the main risk treatments.
* Continuous learning is normal:
  + New scenarios, **edge cases**, upstream model changes, and **dual codebases** (MARS & Vasara) mean you keep encountering fresh details.

**Quick “How-To” (operational checklist)**

1. **Config tab**
   * Set **ValuationDate**.
   * Select **Curves/Surfaces** to load; choose **Scenario** (VaR or named Stress).
2. **Load MD (GXL)**; verify **base objects** appear.
3. **Apply bumps** (function assembles bucket-shift list → builds **bumped curves**).
4. **Pricing tab (per trade)**
   * Confirm **Base NPV** matches source (Calypso/MARS log).
   * Clone formula → **Shifted NPV**, replacing MD refs with **bumped** objects.
   * Compute **Stressed P&L** = Shifted − Base.
5. **Portfolio aggregation** (outside this sheet).

**Notes, tips, and gotchas**

* Keep **GMI tabs** per product type; they’re **stable** (ETD instruments are standardized).
* For **Calypso** tabs, don’t try to be universal—**clone** and **lightly customize** per position family.
* Always cross-check **day count**, **compounding**, **collateral discounting**, **FX treatment**, and **curve mapping** (OIS vs IBOR) before blaming scenarios.
* For VaR: ensure you’re pulling **t vs t-1** correctly from the **same time-series buckets** used in production.

**Mini-glossary**

* **Full Reval**: Reprice with **actual scenario-bumped** market data (no Taylor approximation).
* **Delta-Gamma**: Approximate P&L with first/second-order Greeks from small bumps.
* **Bumped Curve**: A single curve object where **all buckets** reflect the scenario’s shifts.
* **GXL**: Loader interface for trades/market data (Calypso/GMI/etc.).
* **Calypso/GMI/Broadridge**: Systems of record/pricing sources by product family.
* **EMR/MARS/Vasara**: Internal stacks for market data, risk, and pricing context/harness.

**One-slide summary (talk track)**

* **Goal**: Full Reval = **Load MD → Bump MD → Reprice → P&L** (Bumped − Base).
* **Where**: Drive from **Configuration**; generate **bumped curves** from scenario tables/time series; price on **product tabs**.
* **Why**: Captures **true non-linearities** vs delta-gamma.
* **Reality**: Calypso trades vary—**ad-hoc tabs** per family; **GMI** has stable product tabs.
* **Debug**: Use **run-context harness**; step through code to verify inputs/conventions.
* **Build vs Buy**: Python replication is **feasible but large**—you’re rebuilding **plumbing** similar to Vasara/EMR.

If you want, I can convert this into a printable 1–2 page SOP or a deck outline with speaker notes.

Excellent catch 👍 — this gets to the **heart of how full reval works**.

**Why we *don’t* shock the bond price directly**

* The **bond price (99)** is the *output* of a pricing function.
* What actually drives the bond price are **inputs**: discount factors, forward curves, spreads, volatility, etc.
* In stress testing or VaR, we never say “let’s assume the bond falls to 94.” Instead, we say “let’s shock the market environment” (interest rates + spreads, etc.), and then **recalculate** the bond price.
* This way the re-pricing respects the product’s payoff structure and sensitivity.

**Analogy**

Imagine you’re testing a recipe:

* The **cake** (bond price) is the *final product*.
* To test stress, you don’t magically shrink the cake by 5%. You **change the ingredients** (flour, sugar, oven temp = curves, spreads) and then bake again to see how much smaller the cake comes out.

**Why this matters**

* **Consistency across portfolio:** If we shocked prices directly, different instruments (bonds, CDS, swaps) wouldn’t stay internally consistent — e.g., bond falls but CDS doesn’t move in sync.
* **Risk attribution:** By shocking market factors (treasury, spreads, vols), we can explain losses as “X from rates, Y from spreads, Z from FX.”
* **Regulatory standards:** Basel rules for VaR/SVaR require shocks to **risk factors**, not outputs.

So:

* **Full reval:** change the *inputs* (market curves, spreads) → re-price trade → bond price changes as a result.
* **Bond price itself is not shocked** because that would skip the “how” and remove the link between risk factors and instruments.

👉 Want me to illustrate with a **side-by-side table**: “Shock the bond price directly vs shock inputs (treasury + spreads)” to show why the second method is correct for risk?

Perfect — let’s do it in **two voices** 🎓

**🧒 To a 5th grader**

Imagine you own a toy car.

* If I want to know how the car behaves on a bumpy road, I can actually **drive it on the road** and see what happens. That’s **full revaluation** — we try it for real, every time.
* Or, I can just **push the car a little in my hand**, see how it moves, and **guess** what will happen on a bigger road. That’s **delta–gamma** — I don’t really drive it, I just use what I know from small pushes to estimate.

👉 Full reval is the *real test*. Delta–gamma is a *shortcut guess*. Sometimes the guess is good, but if the road is very twisty, the guess can be wrong.

**🎓 To a high schooler**

Think of it like testing how far a basketball will roll.

* **Full revaluation:** You **actually roll the ball** down the hill for each slope and measure the distance. It’s accurate but time-consuming.
* **Delta–gamma:** Instead, you take physics formulas:
  + *Delta* is like “the first slope effect” (linear term).
  + *Gamma* is like “the curve of the hill” (second-order effect).  
    You plug the slope change into those formulas to **approximate** how far the ball goes. Faster, but only works well if the slope change is small and the hill is smooth.

👉 In finance:

* Full reval = pricing each trade under shocked market data.
* Delta–gamma = using stored sensitivities (first & second derivatives) to approximate the change.

Would you like me to **layer the explanation** so you can use it in a meeting (5th grade → high school → professional), like a 3-step elevator pitch? That way you can start simple and build sophistication.

Gotcha—let’s make it concrete and side-by-side.

**Inputs you need**

**Full Revaluation**

* Base market data (curves, vols, FX, spreads, surfaces).
* **Shocked** market data for each scenario.
* Trade static + pricing models.
* Compute price under base and under each shocked state → PnL = shocked − base.

**Delta–Gamma Approximation**

* Base price.
* First-order Greeks (**Δ**) for each risk factor.
* Second-order Greeks (**Γ**): diagonal (curvatures) and cross terms (interaction between factors).
* Shock vector **Δx** for each factor.
* Approximate PnL:

ΔV≈∑iΔi Δxi  +  12∑iΓii(Δxi)2  +  ∑i<jΓij Δxi Δxj\Delta V \approx \sum\_i \Delta\_i\,\Delta x\_i \;+\; \tfrac12\sum\_i \Gamma\_{ii}(\Delta x\_i)^2 \;+\; \sum\_{i<j}\Gamma\_{ij}\,\Delta x\_i\,\Delta x\_j

**Example A — Single factor (showing match vs. miss)**

Think of one “spread” factor xx. Use a toy price function to avoid model jargon.

**A1. Quadratic instrument (no higher orders)**

Price under factor xx:  
V(x)=100+5x−2x2V(x) = 100 + 5x - 2x^2

* Base: x=0⇒V(0)=100x=0 \Rightarrow V(0)=100
* Greeks at base:
  + Δ=dVdx∣0=5\Delta = \frac{dV}{dx}\big|\_{0} = 5
  + Γ=d2Vdx2∣0=−4\Gamma = \frac{d^2V}{dx^2}\big|\_{0} = -4

**Shock 1: x=+0.10x=+0.10**

* **Full Reval**:  
  V(0.10)=100+5(0.10)−2(0.10)2=100+0.5−0.02=100.48V(0.10)=100 + 5(0.10) - 2(0.10)^2 = 100 + 0.5 - 0.02 = 100.48  
  ⇒ΔV=0.48\Rightarrow \Delta V = 0.48
* **Delta–Gamma**:  
  ΔV≈Δx⋅Δ+12Γ(Δx)2=0.10⋅5+12(−4)(0.10)2=0.5−0.02=0.48\Delta V \approx \Delta x\cdot \Delta + \tfrac12\Gamma (\Delta x)^2 = 0.10\cdot 5 + \tfrac12(-4)(0.10)^2 = 0.5 - 0.02 = 0.48

**Result:** Perfect match (because the true function is quadratic and delta–gamma is exact up to second order).

**Shock 2: x=+0.50x=+0.50**

* **Full Reval**:  
  V(0.50)=100+2.5−0.5=102.0⇒ΔV=2.0V(0.50)=100 + 2.5 - 0.5 = 102.0 \Rightarrow \Delta V = 2.0
* **Delta–Gamma**:  
  ΔV≈0.5⋅5+12(−4)(0.5)2=2.5−0.5=2.0\Delta V \approx 0.5\cdot 5 + \tfrac12(-4)(0.5)^2 = 2.5 - 0.5 = 2.0

**Result:** Still an exact match (still quadratic).

**A2. Add a cubic term (optional/vol effects)**

Now let V(x)=100+5x−2x2+1.2x3V(x)=100 + 5x - 2x^2 + 1.2x^3 (weak nonlinearity beyond gamma).

* Base Greeks at x=0x=0: Δ=5,  Γ=−4\Delta=5,\; \Gamma=-4 (same as before; the cubic only appears in higher orders)

**Shock 1: x=+0.10x=+0.10**

* **Full Reval**:  
  Extra cubic = 1.2(0.10)3=1.2×0.001=0.00121.2(0.10)^3 = 1.2 \times 0.001 = 0.0012  
  So ΔV=0.48+0.0012=0.4812 \Delta V = 0.48 + 0.0012 = 0.4812
* **Delta–Gamma**: 0.480.48

**Error:** 0.00120.0012 (tiny for small shocks).

**Shock 2: x=+0.50x=+0.50**

* **Full Reval**:  
  Cubic = 1.2(0.5)3=1.2×0.125=0.151.2(0.5)^3 = 1.2 \times 0.125 = 0.15  
  So ΔV=2.0+0.15=2.15 \Delta V = 2.0 + 0.15 = 2.15
* **Delta–Gamma**: 2.02.0

**Error:** 0.150.15 (now visible).  
**Moral:** Larger shocks or higher-order effects (path dependency, barriers, callable features, vol-of-vol) break the delta–gamma approximation.

**Example B — Two factors with cross-gamma**

Let two factors be xx (credit spread) and yy (interest rate).  
Price:

V(x,y)=100+4x−3y−1.5x2+0.5y2+2xyV(x,y)=100 + 4x - 3y - 1.5x^2 + 0.5y^2 + 2xy

* Base: V(0,0)=100V(0,0)=100
* Greeks at base:
  + Δx=4,  Δy=−3\Delta\_x=4,\; \Delta\_y=-3
  + Γxx=−3,  Γyy=1,  Γxy=2\Gamma\_{xx}=-3,\; \Gamma\_{yy}=1,\; \Gamma\_{xy}=2

**Shock: x=+0.20,  y=−0.30x=+0.20,\; y=-0.30**

**Full Reval**

* Linear: 4(0.20)−3(−0.30)=0.8+0.9=1.74(0.20) - 3(-0.30) = 0.8 + 0.9 = 1.7
* Quadratics: −1.5(0.20)2=−1.5(0.04)=−0.06-1.5(0.20)^2 = -1.5(0.04) = -0.06; +0.5(−0.30)2=0.5(0.09)=0.045+0.5(-0.30)^2 = 0.5(0.09)=0.045
* Cross: 2(0.20)(−0.30)=−0.122(0.20)(-0.30) = -0.12
* Total change: 1.7−0.06+0.045−0.12=1.5651.7 - 0.06 + 0.045 - 0.12 = 1.565

**Delta–Gamma**

ΔV≈ΔxΔx+ΔyΔy+12Γxx(Δx)2+12Γyy(Δy)2+ΓxyΔxΔy=4(0.20)+(−3)(−0.30)+12(−3)(0.04)+12(1)(0.09)+2(0.20)(−0.30)=0.8+0.9−0.06/2+0.09/2−0.12=1.7−0.03+0.045−0.12=1.565\begin{aligned} \Delta V &\approx \Delta\_x \Delta x + \Delta\_y \Delta y + \tfrac12\Gamma\_{xx}(\Delta x)^2 + \tfrac12\Gamma\_{yy}(\Delta y)^2 + \Gamma\_{xy}\Delta x \Delta y \\ &= 4(0.20) + (-3)(-0.30) + \tfrac12(-3)(0.04) + \tfrac12(1)(0.09) + 2(0.20)(-0.30) \\ &= 0.8 + 0.9 - 0.06/2 + 0.09/2 - 0.12 \\ &= 1.7 - 0.03 + 0.045 - 0.12 \\ &= 1.565 \end{aligned}

**Result:** Exact match (it’s purely quadratic).  
If we add a cubic piece (e.g., +0.8x3+0.8x^3), **Full Reval** would be 1.565+0.0064=1.57141.565 + 0.0064 = 1.5714, while **Delta–Gamma** stays 1.5651.565 → small gap that grows with shock size.

**Quick cheat sheet**

* **When they match:**  
  Small shocks, instruments well-approximated by a quadratic in the shocked factors, or portfolios dominated by linear/convex exposures without jumps/path dependency.
* **When they diverge:**  
  Big shocks, strong optionality/callability/barriers, vol changes (vega/vanna/volga), basis/nonparallel curve moves, discrete features (calls/caps/floors), path dependency.
* **Inputs difference in practice:**
  + *Full Reval:* scenario → rebuild **all** market objects → re-price trades.
  + *Delta–Gamma:* scenario → apply shocks to **factors**, multiply by stored **Greeks** (and cross-Greeks), sum up.

If you want, I can plug in your **actual factor buckets** (e.g., IR 2s/5s/10s/30s nodes, CDS tenors, EQ vols) and show a miniature portfolio where we compute both numbers step-by-step exactly like above.

That’s a very sharp question 🔑 — and you’re not wrong, it depends on **what role you’re playing**:

**1. Two different “worlds”**

**a. Market pricing / calibration world**

* Here, the **observed bond price (say 99)** **is the input**.
* You feed that into your system to **calibrate unobservable things**:
  + OAS (spread over Treasuries)
  + Implied default probability
  + Recovery assumptions, etc.
* So in this sense: **bond price is indeed an input** to solve for the model parameters that make PV(model) = PV(market).

👉 Every day, your system ingests the market bond price and backs out the spread/OAS.

**b. Risk / simulation world**

* Once you’ve calibrated those curves/spreads from market prices, the **price becomes an output** of your risk engine.
* Why? Because for scenarios, you cannot “shock the bond price” in isolation — it would break consistency.
* Instead, you shock **the underlying risk drivers** (rates, spreads, default intensity). Then you re-price the bond using your model.

👉 That’s why in full reval, we shock Treasuries + spreads, not the bond price directly.

**2. What if bond price falls to 60 or 20?**

This is the credit blow-up case you’re imagining.

* If your **discounting factor model (Treasury + sector spread)** can’t explain it, then:
  + You need **bigger spreads** → your calibration step would solve for a much wider OAS.
  + Or you need a **different model** (e.g., survival-based credit model with hazard rates).

So:

* Market price 60 → feed into calibration → back out implied spread maybe 3000 bps.
* Then for risk, that spread curve is now your input.

**3. Analogy**

* **Calibration step**: Doctor measures your blood pressure (market bond price) → converts it into a health index (spread/hazard).
* **Scenario step**: To see how you’ll feel tomorrow, the doctor doesn’t randomly lower your health score. He **changes diet/salt intake (risk drivers)** and uses the model to predict the new score.

✅ **So you are correct:** bond price *is* an input at the calibration stage.  
But once calibrated, in risk engines (VaR, stress), **we don’t shock prices directly** — we shock the drivers and re-price.

Do you want me to show you a **numerical illustration** of this dual role? (e.g., “Bond at 99 today → calibrate spread. Bond collapses to 60 → recalibrate spread. Then show how the spread, not the price, is shocked in scenarios.”)