A hierarchical (multi‐level) approach for generating the parameters of each “knowledge product” is definitely a good way to capture variation across the portfolio while maintaining some global structure. You can, for example, specify hyperpriors for:

• Utility process parameters (e.g. κ, σ)

• Shock frequency and severity parameters (λ, shock\_σ, shock\_scale)

• Starting truths (T₀), replaceability factors, etc.

Then, for each knowledge product you would draw from these “upper-level” (hyper) distributions to get a set of parameters that go into the existing code blocks (calculate\_kdg\_payout or similar). This setup allows you to reflect that some knowledge products are “similar” (e.g. in the same field, with similar shock frequencies and volatilities), while others differ more substantially.

-------------------------------------------------------------------------------

HOW TO INTRODUCE TAIL CORRELATION

-------------------------------------------------------------------------------

Right now, your simulation code typically draws random shocks, truth levels, etc. independently per product. If you want to introduce correlated tail events (e.g. a “meta-shock” that degrades multiple knowledge products at once), you have two main paths:

1) Keep the same functions but feed correlated random draws.

2) Rewire the functions to compute portfolio outcomes from a single global simulation loop.

-------------------------------------------------------------------------------

1) Feeding Correlated Random Draws

-------------------------------------------------------------------------------

• At the top level, generate correlated sequences of random variables for each shock or drift component. For instance, you could draw from a multivariate distribution or use a copula (Student‐t, Gumbel, etc.) to produce an n×d array of correlated shocks, where

– n is the number of simulation runs,

– d is the number of knowledge products (or the dimension of random variables needed).

• Pass each product’s slice of the random draws to sim\_truth\_level or util simulations instead of letting them call numpy/scipy’s RNG internally on their own. That way, all products share a portion of the same random vectors in each iteration, guaranteeing correlation.

• In practice, you would rework sim\_truth\_level, for example, to accept external arrays for “erosion draws,” “shock draws,” etc. The function would no longer compute np.random.randn(...) itself but would consume an array slice that you’ve pre‐generated in correlated fashion.

-------------------------------------------------------------------------------

2) A Single Global Simulation Loop

-------------------------------------------------------------------------------

• Another approach is to write a higher‐level “simulate\_portfolio(...)” function where, at each step in time (or each iteration), you draw the “global” random factors (shocks, drifts, etc.) from a correlated distribution, then apply them to each product’s state transitions in a single loop.

• If the correlation structure is primarily about a shared “shock arrival,” you can generate one shock event for the entire portfolio (e.g. a “replication crisis”) and apply it at once to all knowledge products, with product‐specific severities or offsets if necessary.

Both strategies require some level of rewiring. You don’t necessarily have to replace your existing code entirely, but you do need to allow the random draws to be externally injected rather than generated independently in each function call.

-------------------------------------------------------------------------------

WRAPPING UP

-------------------------------------------------------------------------------

• Hierarchical parameter draws let you capture how individual products’ parameters vary around some global means/variances.

• Correlated tail events require coupling the random draws used by each product.

• Whether you feed correlated draws into the existing code or write a new portfolio‐wide simulator depends on how much you want to reuse the current structure.

In short:

1. Yes, a hierarchical model for hyperparameters is an excellent choice to handle “portfolio‐level” variation.

2. Introducing tail correlation does require at least partial rewiring so that each knowledge product doesn’t run purely independently. You can still reuse most of your functions, but you must supply the correlated draws (or a correlated shock factor) from a higher level.