

```

# VCF-RESEARCH — /code Directory Architecture
## Overview
This document defines the final implementation layout for the **VCF-RESEARCH/code** directory.
It consolidates all math engines, utilities, and core modules into a clean, modern structure that
Claude (and other AI collaborators) can work within efficiently.

---
# Final Directory Structure
```
VCF-RESEARCH/
code/
regime_engine/
sector_engine/
unified_engine/
wavelit_engine/
utils/
```

Each engine corresponds directly to a research Phase:
- **Phase I:** `regime_engine`
- **Phase II:** `sector_engine`
- **Phase III:** `unified_engine`
- **Phase IV:** `wavelit_engine`
---

# Engine Details
## 1. regime_engine/
Handles economic cycle modeling.
**Modules:**
- `regime_engine.py` – main engine
- `regime_metrics.py` – macro + liquidity metrics
- `normalization.py` – standardization utilities
```
```
## 2. sector_engine/
Handles sector interaction, sector cycles, and sector significance.
**Modules:**
- `sector_engine.py` – main entry point
- `sector_metrics.py` – dispersion, breadth
- `sector_harmonics.py` – FFT/low-level harmonic analysis
```
```
## 3. unified_engine/
Combines Phase I + II into a unified VCF state space.
**Modules:**
- `unified_engine.py`
- `pca_geometry.py`
- `feature_vector.py`
```
```
## 4. wavelit_engine/
CWT, MODWT, resonance analysis, and final VCF regime structure.
**Modules:**
- `wavelit_engine.py`
- `wavelet_math.py`
- `resonance_math.py`
- `wavelet_features.py`
```
```
## 5. utils/

```

Shared tools not tied to any specific engine.

Modules:

- `loaders.py` – load raw/clean data
- `filepaths.py` – manage all paths
- `transforms.py` – helper math functions
- `plotting.py` – visual helpers

Why `/code` is the Correct Design

✓ Simplifies navigation

Claude and other AI tools instantly understand the purpose of `/code`.

✓ Avoids nested engine directories

Keeps all functionality grouped under one conceptual layer.

✓ Matches modern, open-source norms

Researchers expect to find implementation code inside `/code` or `/src`.

✓ Future-proof

New engines can easily be added later:

``

credit_engine/

global_engine/

crypto_engine/

``

✓ Clean separation

- `/code` → implementation
- `/docs` → specifications
- `/pilots` → mathematical test suite
- `/notebooks` → experimentation
- `/data_clean` `/data_raw` → inputs and processed data

Next Recommendations

1. Allow Claude to implement or reorganize code inside `/code` as needed.
2. Once Claude finishes a stable pass, run a **repo analysis** to confirm no drift.
3. Use the Pilot Test Suite to validate each engine independently.
4. Continue documenting major updates inside `/docs`.

End of Document