

# The App's core idea

---

## A) The core philosophy: a 5-plane Sudoku world model

You describe a financial analysis idea as if it were a **5-round, 2D-plane Sudoku**, played **day by day**.

There are **five planes**, each one representing a major layer of the economy:

- Supply
- Manufacturing
- Consumption
- Logistics
- Financial services (and services in general)

Within each plane, there is a **grid like Sudoku**:

- **Each cell** in the grid represents a **company**.
- The **squares inside the cell** represent the company's **internal decision-making variables**.
- A **group of cells** represents a **business sector**, from the smallest to the biggest.

There are also **linked variables** (external decision-making variables). They are tied to companies connected through:

- The same supply chain
- Competitors
- Governments
- Services (financial or not)
- Logistics
- Central banks
- Portfolio relationships

This is presented as the background philosophy for the code you want to make, called **Operator 1**.

---

## B) What “Operator 1” does (high-level behavior)

Operator 1 is meant to:

1. Take a **specific company**.
2. Build a **history cache** (you specify about **2 years**) for:
  - Internal decision variables
  - Linked (external) variables
3. Use a Sudoku-like idea: **infer hidden variables** from the variables that are known.
4. Run a **temporal analysis** using several mathematical modules:
  - Start from the first day in the historical cache.
  - Predict the next day.
  - Compare the prediction with the cache.
  - Iterate day-by-day until reaching the present.
5. Run a **burn-out process** (accuracy improvement loop):
  - Take the pattern result from temporal analysis.
  - Apply it repeatedly on cache data from about **half a year prior to present**.
  - Repeat multiple times to improve accuracy.
6. Output a **PDF report** containing:
  - A company report in a Bloomberg-style format
  - A prediction of the stock chart
  - A candlestick series of patterns expected to happen
  - “All the data a financier would need”
  - A modified style that also includes the model’s linked-variable data

You also state that the final report would be generated by the **Gemini API**.

---

## C) Survival mode logic (company + country)

### C.1) What is Survival Mode?

**Survival mode** is a state where normal business operations shift to **crisis management**. The company (or country) focuses on immediate survival rather than growth or optimization.

**Company Survival Mode** is triggered when:

- Liquidity is critically low (current ratio < 1.0)
- Debt is dangerously high (debt-to-equity > 3.0)
- Cash flow is negative ( $FCF < 0$ )
- Market stress is extreme (drawdown > 40%)

**Country Survival Mode** is triggered when:

- Credit spreads spike (indicating banking stress)
- Currency collapses (FX volatility extreme)
- Unemployment surges (recession signal)
- Yield curve inverts severely (economic distress)
- Government policy rates hit extreme levels

## C.2) How Survival Mode Affects Calculations

**In Normal Mode:**

- The temporal models optimize for **growth** and **profitability**
- Decision variables are weighted toward expansion metrics (revenue growth, market share, R&D)
- Risk tolerance is moderate
- Time horizon is medium to long-term (quarters to years)

**In Survival Mode:**

- The temporal models shift to **preservation** and **liquidity**
- Decision variables are re-weighted using the **Survival Hierarchy** (see below)
- Risk tolerance drops to near-zero
- Time horizon compresses to immediate-term (days to weeks)
- Predictions emphasize downside protection over upside capture

### C.3) The Survival Hierarchy (Priority Ranking)

When survival mode is active, decision variables are re-weighted according to this **strict priority order**:

#### Tier 1: Liquidity & Cash (Highest Priority)

1. `cash_and_equivalents_asof` - Actual cash on hand
2. `cash_ratio` - Can the company pay immediate bills?
3. `free_cash_flow_ttm` - Is the company generating or burning cash?
4. `operating_cash_flow_asof` - Core business cash generation

#### Tier 2: Debt & Solvency

1. `total_debt_asof` - Absolute debt burden
2. `debt_to_equity` - Leverage fragility
3. `net_debt_to_ebitda` - Debt serviceability
4. `interest_coverage` - Can the company afford its interest payments?
5. `current_ratio` - Short-term obligation coverage

#### Tier 3: Market Stability

1. `volatility_21d` - Is the stock in freefall?
2. `drawdown_252d` - How much value has been destroyed?
3. `volume` - Is there still market liquidity?

#### Tier 4: Profitability (Lower Priority in Survival)

1. `gross_margin` - Unit economics
2. `operating_margin` - Operating efficiency
3. `net_margin` - Bottom-line profitability

#### Tier 5: Growth & Valuation (Lowest Priority in Survival)

1. `revenue_asof` - Revenue levels
2. `pe_ratio` - Market valuation (almost irrelevant in crisis)
3. `ev_to_ebitda` - Enterprise valuation

#### Mathematical Implementation:

- In normal mode, all tiers have roughly equal weight (Tier 1: 20%, Tier 2: 20%, Tier 3: 20%, Tier 4: 20%, Tier 5: 20%)
- In survival mode, weights shift dramatically:
  - Tier 1: **50%** (liquidity is everything)
  - Tier 2: **30%** (debt kills companies in crisis)
  - Tier 3: **15%** (market panic signals)
  - Tier 4: **4%** (profitability matters less when survival is at stake)
  - Tier 5: **1%** (growth is irrelevant in survival)

#### C.4) Country Survival Mode Interaction

The system checks **both** company and country survival modes:

##### **Case 1: Normal Company, Normal Country**

- Use standard weights (normal mode)
- No adjustments needed

##### **Case 2: Survival Company, Normal Country**

- Use company survival hierarchy (50/30/15/4/1 weights)
- Company is in distress but environment is stable
- Focus: internal restructuring, liquidity management

##### **Case 3: Normal Company, Survival Country**

- Check if company is **Protected** (see C.5 below)
- If protected: use standard weights (company is shielded)
- If NOT protected: use **modified survival hierarchy** (40/35/20/4/1)
  - Reason: company is healthy but environment is toxic
  - Focus: defensive positioning, preserve value

##### **Case 4: Survival Company, Survival Country**

- Use **extreme survival hierarchy** (60/30/10/0/0)
- Both company and environment are in crisis
- Focus: immediate survival only, ignore profitability and growth entirely

- Tier 1 (liquidity) becomes 60% of the model weight
- Tier 4 and 5 drop to 0% (they don't matter in extreme crisis)

### C.5) Country Protection Flag (World Bank Open Data-based)

When a country enters survival mode, the system checks if the company is **protected** by government policy:

#### **Protection Indicators (from World Bank macro data):**

##### **1. Strategic sector designation**

- Is the company in a sector the government explicitly protects? (defense, energy, banking, utilities)
- If `sector` matches protected list → Protected = True

##### **2. Government intervention & capacity signals (proxy-driven)**

- World Bank is mostly low-frequency macro. Do not assume daily policy facilities exist.
- Use best-effort proxies (config-driven), for example:
  - elevated inflation + deteriorating FX reserves + worsening current account
  - GDP growth deterioration
- If proxies indicate crisis capacity and company is strategic/systemic → Protected = True

##### **3. Market size threshold (GDP from World Bank)**

- Is the company "too big to fail"?
- Fetch `gdp_current_usd` (World Bank indicator) and test:
  - `market_cap > 0.1% of GDP` → Protected = True

#### **Implementation:**

- Store as boolean: `country_protected_flag`
- If protected: insulate from country survival mode (use company's own mode only)
- If not protected: apply country survival mode weights

## C.6) Vanity Expenditure Analysis

### What is Vanity Spending?

Vanity expenditure is spending that **signals status** rather than creating value:

- Excessive executive compensation
- Luxury offices and facilities
- Aggressive marketing without ROI
- Empire-building acquisitions
- Shareholder-unfriendly behavior (buybacks while burning cash)

### Why it matters:

- Companies with high vanity spending are more fragile in downturns
- Vanity spending is often the first thing cut in survival mode
- High vanity percentage predicts lower survival probability

### Vanity Percentage Calculation:

```
vanity_percentage = (vanity_expenditures / revenue_ttm_aso_f) * 100
```

### Components of Vanity Expenditures:

#### 1. Executive compensation excess

- Formula: `exec_comp_excess = max(0, executive_compensation - (net_income_ttm * 0.05))`
- Rationale: Executive comp above 5% of net income is excessive

#### 2. SG&A bloat

- Formula: `sga_bloat = max(0, sga_expenses - (revenue_ttm * industry_median_sga_ratio * 1.2))`
- Rationale: SG&A more than 20% above industry median is wasteful

#### 3. Negative FCF with buybacks

- Formula: `vanity_buybacks = max(0, stock_buybacks) if fcf_ttm < 0 else 0`
- Rationale: Buying back stock while burning cash is vanity

#### 4. Marketing excess in distress

- Formula: `marketing_excess = max(0, marketing_expenses - (revenue_ttm * 0.10))  
if company_survival_mode else 0`
- Rationale: Marketing above 10% of revenue during distress is vanity

### Total Vanity Expenditure:

```
vanity_expenditures = exec_comp_excess + sga_bloat + vanity
                      _buybacks + marketing_excess

vanity_percentage = (vanity_expenditures / revenue_ttm_aso
                     f) × 100
```

### Vanity Percentage Interpretation:

- **0-2%**: Disciplined, lean operations
- **2-5%**: Normal, acceptable level
- **5-10%**: Elevated, watch for cuts
- **10-20%**: High, company is bloated
- **>20%**: Extreme, survival probability is low

### How Vanity Percentage Affects Survival Analysis:

- Companies with `vanity_percentage > 10%` get a **survival penalty**
- In survival mode, high vanity companies are predicted to:
  - Cut expenses aggressively (which can cause operational disruption)
  - Face shareholder pressure
  - Have lower probability of recovery
- In the temporal models, vanity percentage **reduces the weight on Tier 4 and Tier 5 variables even further**

### Integration into Survival Hierarchy:

- If `vanity_percentage > 10%` and `company_survival_mode == True`:
  - Apply additional weight shift: move 2% from Tier 4 and 3% from Tier 5 to Tier 1
  - New weights become: 55/30/15/2/0 (even more emphasis on liquidity)

## C.7) Summary: Survival Mode Logic Flow

**Step 1:** Compute company survival indicators

→ Set `company_survival_mode_flag` (True/False)

**Step 2:** Compute country survival indicators from World Bank Open Data (as-of aligned)

→ Set `country_survival_mode_flag` (True/False)

**Step 3:** If country survival mode, check protection

→ Set `country_protected_flag` (True/False)

**Step 4:** Compute vanity percentage

→ Store `vanity_percentage` (0-100)

**Step 5:** Select hierarchy weights based on state:

- Normal/Normal → Standard weights (20/20/20/20/20)
- Survival Company/Normal Country → Company survival (50/30/15/4/1)
- Normal Company/Survival Country (not protected) → Modified survival (40/35/20/4/1)
- Normal Company/Survival Country (protected) → Standard weights (20/20/20/20/20)
- Survival Company/Survival Country → Extreme survival (60/30/10/0/0)
- If vanity > 10% in survival → Shift +5% to Tier 1 from Tier 4/5

**Step 6:** Pass the selected hierarchy weights to temporal analysis modules

**Step 7:** Temporal models use these weights when:

- Predicting next-day variables
- Computing feature importance
- Running burn-out optimization
- Generating pattern predictions

---

## D) Data sources and flow (Eulerpool + World Bank Open Data + FMP for OHLCV)

Operator 1 uses:

- **Eulerpool** for company identity and fundamentals (profile, statements, relationships, etc.).

- **World Bank Open Data** for country-level macro variables.
- **FMP (Financial Modeling Prep)** specifically to fill the **OHLCV** gap when Eulerpool does not provide a usable OHLCV time series.

**Why World Bank:** global coverage, consistent country identifiers, and broad macro breadth.

**Why FMP:** reliable daily and intraday market candles (OHLCV), so the 2-year cache always contains the market series required by downstream feature engineering and chart/pattern modules.

### D.1) Company resolution workflow (User Input → ISIN → Eulerpool)

Operator 1 can resolve a company without relying on an Eulerpool “search” endpoint.

**Minimum Eulerpool endpoints (strict):**

1. `GET /api/1/equity/profile/{isin}` — identity + `country`, `currency`, `sector`, `industry`, shares/mcap when available.
2. `GET /api/1/equity/quotes/{identifier}` — 2-year daily market series (OHLCV is still pending confirmation).
3. `GET /api/1/equity/incomestatement/{isin}` — fundamentals.
4. `GET /api/1/equity/balancesheet/{isin}` — solvency + liquidity.
5. `GET /api/1/equity/cashflowstatement/{isin}` — cash reality.
6. `GET /api/1/equity/peers/{isin}` — seed competitor/peer set.
7. `GET /api/1/equity/supply-chain/{isin}` — seed suppliers/customers.
8. `GET /api/1/equity/executives/{isin}` — exec compensation + tenure (vanity + stability).

\*(For each linked ISIN discovered via peers/supply-chain, repeat only the required endpoints with a budget cap.)

**Goal:** Convert user input (name + optional country/ticker) into a verified `target_isin`.

**Recommended flow (Gemini as bridge):**

1. User provides: `company_name`, optional `company_country`, optional `company_hint_ticker`.

2. Gemini returns:

- candidate identifiers: `isin`, `ticker`, exchange hints
- a source link or rationale.

3. **Verification step (required):** call `GET /api/1/equity/profile/{isin}` and confirm:

- profile `ticker` matches expected (when available)
- profile `name` is similar to user input
- profile `country` is consistent (when user provided a hint)

4. If verification fails, ask Gemini for the next-best candidate and repeat.

**Reliability rule:** never accept an ISIN from Gemini unless it is validated by Eulerpool's profile endpoint.

\*(After ISIN is verified, the rest of the pipeline remains unchanged.)

## D.2) Macro workflow (Country → World Bank indicators)

Once the company is confirmed, extract `resolved_country` from Eulerpool profile.

### Canonical macro variables (World Bank-backed) the system must support

These replace the previous FRED canonical variables without changing the core idea (macro-linked variables that drive country survival mode, protection, and real-return adjustments).

#### A) Purchasing power / inflation (required)

- `inflation_rate_yoy` (annual %)
- `cpi_index` (index)
- Derived:
  - `inflation_rate_daily_equivalent` (derived from YoY)
  - `real_return_1d = nominal_return_1d - inflation_rate_daily_equivalent`

#### B) Rates & monetary conditions (best-effort / proxy)

World Bank does not provide a universal daily policy-rate series. Use these as canonical substitutes (or keep missing flags if not available):

- `real_interest_rate`
- `lending_interest_rate`

- `deposit_interest_rate`
- `inflation_rate_yoy` (already above) as the tightening signal proxy
- If you later add a secondary rates provider, keep the canonical names but leave the World Bank module as a fallback.

### C) FX and external stress (recommended)

- `official_exchange_rate_lcu_per_usd`
- `reserves_months_of_imports`
- `current_account_balance_pct_gdp`

### D) Real economy health (recommended)

- `unemployment_rate`
- `gdp_growth`
- `gdp_current_usd`
- `gdp_per_capita_current_usd`
- `industrial_value_added_pct_gdp` (proxy for industrial health)

## Implementation requirements

1. **Config-driven indicator mapping** in `config/world_bank_indicator_map.yml`:
  - Maps canonical variable name → World Bank indicator code.
  - Allows `null` when no good indicator exists.
2. **Country mapping** uses ISO-2 from Eulerpool when possible, but World Bank often prefers ISO-3.
  - Implement `iso2 -> iso3` conversion using World Bank country endpoint (cached).
3. **Frequency alignment to daily cache:**
  - Most World Bank indicators are annual (some are monthly).
  - Align to daily using **as-of** logic exactly like statements: attach the last published value with `value_date <= t`.
  - Also store `macro_asof_date_<var>` and `is_stale_<var>` flags (e.g., stale if > 365 days old).

4. **Missingness policy remains:** value `null` + `is_missing_<var> = 1`, and continue.

### D.3) Complete data source flow summary (updated)

1. **User Input** → Gemini generates search plan → **Eulerpool search** → user confirms → target company resolved
  2. **Target company country** → **World Bank indicator fetch** → macro variables loaded (as-of aligned)
  3. **Target company** → Gemini discovers relationships → Eulerpool searches → linked entities resolved
  4. **Target + linked entities** → fetch decision variables from Eulerpool
  5. **World Bank macro + decision variables** → compute survival modes
  6. Continue to temporal analysis...
- 

### D.4) API reliability requirements (timeouts, retries, rate limits)

All external calls (Eulerpool, Gemini, World Bank, FMP) must use a shared HTTP layer with:

- Timeouts (connect + read)
- Retries only on: **429, 500, 502, 503, 504**
- Exponential backoff with jitter
- Respect `Retry-After` when present
- Cap retries so the notebook cannot hang indefinitely
- A simple token-bucket rate limiter per host
- Local disk caching of raw API responses (Kaggle) to avoid repeated calls

#### Failure behavior (mandatory):

- **Fail fast** on critical verification steps:
  - target identity verification (Eulerpool profile)
  - target OHLCV verification (FMP quote + ability to download candles)
- **Degrade gracefully** on optional workloads:
  - linked entities discovery and bulk download

- charts, heavy models, non-critical enrichment

Store request metadata in `cache/request_log.json`:

- endpoint, params hash, timestamp, status, latency, retries, error (if any)
- 

## J) Estimating unknown daily variables (replace nulls with point-in-time estimates)

### J.0) Numerical safety (division-by-zero / invalid math)

Many derived variables are ratios. The pipeline must never crash due to division-by-zero or invalid denominators.

**Global rule:** Every derived ratio must be computed with a safe helper that returns:

- a numeric value when valid
- otherwise `null` and sets `is_missing_<var> = 1`
- and logs an `invalid_math_<var>` flag for debugging

#### Example: safe debt-to-equity

- Denominator: `total_equity_asof`
- If `total_equity_asof` is `null`, `0`, or near-zero ( $|x| < \text{epsilon}$ ):
  - set `debt_to_equity = null`
  - set `is_missing_debt_to_equity = 1`
  - set `invalid_math_debt_to_equity = 1`
- Else compute normally.

#### Negative equity handling (common in distress):

- If equity is negative, do not treat it as invalid automatically.
- Store:
  - `debt_to_equity_signed = total_debt_asof / total_equity_asof` (may be negative)
  - `debt_to_equity_abs = total_debt_asof / abs(total_equity_asof)` (always positive when equity  $\neq 0$ )
- Use `debt_to_equity_abs` for threshold-based survival rules.
- Keep the signed version for interpretability.

## Other high-risk ratios (apply same safety):

- `current_ratio = current_assets_asof / current_liabilities_asof`
- `cash_ratio = cash_and_equivalents_asof / current_liabilities_asof`
- `fcf_yield = free_cash_flow_ttm_asof / market_cap`
- `interest_coverage = ebit_ttm_asof / interest_expense_ttm_asof` (handle `interest_expense` near 0)

## Epsilon guidance:

- Use `epsilon = 1e-9` in normalized units, or a unit-aware epsilon after scaling.
  - Always compute on normalized units if possible.
- 

## J.1) Principle: never erase truth

Operator 1's Sudoku philosophy requires **estimating unknown variables** on every day in the cache.

### J.1) Principle: never erase truth

For each variable `x` on day `t`, store **three layers**:

- `x_observed` (raw / computed from raw; may be null)
- `x_estimated` (model-imputed)
- `x_final` (chosen value used by downstream models)

Also store:

- `x_source` in { `observed`, `estimated` }
- `x_confidence` in [0, 1]

Rule:

- If `observed` exists, `x_final = x_observed`.
- Else, `x_final = x_estimated`.

## J.2) Estimator design (point-in-time, regime-aware)

Use a **two-stage imputation** that is fast and consistent:

**Stage 1: accounting-identity inference (deterministic)**

Examples:

- If `total_debt_asof` missing but `short_term_debt_asof` and `long_term_debt_asof` exist:
  - `total_debt_asof = short + long`
- If `enterprise_value` missing but components exist:
  - `EV = market_cap + total_debt_asof - cash_and_equivalents_asof`
- If a ratio missing but numerator/denominator exist:
  - compute ratio directly

## Stage 2: statistical / ML imputation (probabilistic)

For remaining missing values on day `t`:

- Build a feature vector from the **same day** and a small **historical window** ending at `t`.
- Use a regime-aware window:
  - Select historical samples from the **same regime label** as day `t` (or highest-prob regime if soft assignment).
  - If insufficient samples, back off to recent data regardless of regime.

Recommended base model for Kaggle practicality:

- `IterativeImputer` (sklearn) with `BayesianRidge` or `RandomForestRegressor`
- plus a fast fallback: KNN imputer

**Hard constraint:** no look-ahead. Only use data with index `<= t`.

### J.3) How imputation integrates into learning

- The temporal modules must learn on `*_final` values.
- The loss function must penalize errors on observed values more than on estimated ones:
  - `loss = w_obs * error(observed dims) + w_est * error(estimated dims)`
  - where `w_obs > w_est` (e.g., 1.0 vs 0.3)

---

## K) Regime switching redesign (more sophisticated than simple weight switching)

The current hierarchy weight switching is useful but too coarse.

### K.1) Separate regime layers

Maintain **two regime systems**:

1. **Market regime** (return/volatility driven): bull, bear, high-vol, low-vol, crisis.
2. **Fundamental regime** (liquidity/solvency/cash driven): healthy, stressed, survival.

Store both as:

- `regime_market_label`, `regime_market_probs[]`
- `regime_fund_label`, `regime_fund_probs[]`

### K.2) Soft switching instead of hard switching

Instead of picking one regime, compute predictions as a mixture:

- $\text{pred} = \sum_r p(r|t) * \text{pred}_r$

Where each regime has its own model parameters (or its own lightweight model instance).

### K.3) Efficient computation design

To keep compute feasible in Kaggle:

- Pre-train / fit **regime detectors** once per forward pass segment.
- Maintain per-regime rolling sufficient statistics (means, covariances, residual variances).
- Update regime-specific models incrementally rather than refit from scratch.

### K.4) Hierarchy weights become a downstream effect

Keep the 5-tier hierarchy, but compute tier weights as:

- base hierarchy (normal/survival)
- adjusted by **market-regime risk multiplier** (e.g., high-vol shifts weight toward Tier 1-3)
- adjusted by vanity

This yields continuous, interpretable weights rather than discrete jumps.

---

## L) Burn-out redesign → continuous learning with regime-weighted windows

Replace the current burn-out (repeat last 6 months with the same pattern) with a **continuous learning loop**.

### L.1) Regime-weighted historical windows

At any day  $t$  define a training window distribution over past days  $\tau \leq t$ :

- Higher weight for days with similar regimes:
  - $w(\tau) \propto \exp(-\Delta t/half\_life) * similarity(regime(\tau), regime(t))$
- This makes the model remember what matters for the current state.

### L.2) Online evaluation inside training (no external validation required)

For every step in temporal analysis and burn-out:

1. Predict  $t+1$  from state at  $t$ .
2. Compare to cache at  $t+1$ .
3. Update model parameters (online learning).
4. Log errors by tier, by regime, and by observed vs estimated.

This is the **self-contained validation** loop:

- The cache is the ground truth.
- No external split is required to learn and improve.

### L.3) Convergence and safety

Stop continuous burn-out when:

- Regime-weighted rolling error stops improving for N steps, or
  - Model drift exceeds limits (e.g., parameters explode), triggering a reset.
- 

## M) Guarantee: prediction always reconciles with cache

Whether during temporal analysis or burn-out:

- The system must always do **predict** → **compare** → **update**.
- Every module must expose:
  - `predict(state_t)`
  - `update(state_t, actual_t_plus_1, weights)`
- The ensemble must update its weights using regime-weighted recent accuracy.

### D.1) Gemini Workflow for Company Resolution (User Input → Eulerpool)

When the user provides a company name, the system must use Gemini to intelligently search Eulerpool without any hardcoded data.

**Input from user:**

- `company_name` (e.g., "Apple", "Tesla Inc", "Saudi Aramco")
- `company_country` (optional hint, e.g., "USA", "Saudi Arabia", "unknown")
- `company_hint_ticker` (optional, e.g., "AAPL")

**Gemini's role (Phase 1: Search Plan Generation):**

Gemini receives a structured prompt and must output a JSON search plan:

```
{
  "primary_search_terms": [
    "Apple Inc",
    "Apple Computer",
    "AAPL"
  ],
  "alternative_spellings": [
    "Apple Incorporated",
    "Apple Computer Inc"
  ],
  "likely_country_codes": ["US", "USA"],
  "likely_ticker_symbols": ["AAPL", "AAPL.O", "AAPL.NAS"],
  "sector_hints": ["Technology", "Consumer Electronics"],
  "search_strategy": "exact_match_first_then_fuzzy"
}
```

**Gemini prompt template for company resolution:**

You are a financial data search assistant. A user wants to find a company in the Eulerpool financial database.

User provided:

- Company name: "{company\_name}"
- Country hint: "{company\_country}" (may be "unknown")
- Ticker hint: "{company\_hint\_ticker}" (may be empty)

Your task: Generate a comprehensive search plan to find this company in Eulerpool.

IMPORTANT RULES:

1. Do NOT assume the company exists or use prior knowledge about specific companies
2. Generate search variations that would work for ANY company name globally
3. Consider international naming conventions (e.g., "Inc", "Ltd", "GmbH", "SA", "公司")
4. If country is "unknown", suggest multiple likely countries based on company name patterns
5. Output ONLY valid JSON, no additional text

Output JSON schema:

```
{  
    "primary_search_terms": [list of 3-5 search strings to try first],  
    "alternative_spellings": [list of 2-4 alternative name formats],  
    "likely_country_codes": [list of 1-3 ISO-2 country codes, or ["GLOBAL"] if truly unknown],  
    "likely_ticker_symbols": [list of possible ticker formats, or [] if unknown],  
    "sector_hints": [list of 1-3 possible sectors based on name, or [] if unclear],  
    "search_strategy": "exact_match_first_then_fuzzy" or "fuzzy_from_start"  
}
```

Generate the search plan now.

### Code execution after Gemini response:

1. Parse Gemini's JSON output
2. Execute Eulerpool searches in priority order:
  - Try `primary_search_terms` first with exact match
  - If no results, try `alternative_spellings`
  - If still no results, try fuzzy search with primary terms
3. For each Eulerpool result, extract:
  - Company name (official)
  - ISIN
  - Ticker
  - Exchange
  - Country
  - Sector/Industry
4. **Scoring algorithm** (no hardcoded weights, proportional logic):

```
score = 0
if ticker matches user hint exactly: score += 50
if country matches user hint: score += 30
if name similarity > 0.8 (Levenshtein): score += 20
if sector in Gemini's sector_hints: score += 10
```

5. Rank results by score
6. Present top 3-5 matches to user for confirmation
7. User selects correct match
8. Store confirmed identifiers for pipeline

### Country enrichment (World Bank Documents & Reports API):

Once the company is confirmed, extract `resolved_country` from the Eulerpool profile.

## World Bank Documents & Reports API (WDS) role:

Use the World Bank Documents & Reports search API to discover *country-specific* public documents that can support qualitative signals and context (policy, macro stress, sector conditions) without hardcoding sources.

Base endpoint:

- <https://search.worldbank.org/api/v3/wds>

Key query mechanics:

- Broad search across core fields using `qterm` (title, abstract, report number, project metadata, country/region, etc.).
- Restrict to specific fields using query parameters like `count_exact`, `lang_exact`, `docty_exact`, `display_title`, etc.
- Request returned fields with `fl` (comma-separated).
- Paging with `rows` (page size) and `os` (offset).
- Facets with `fct`.
- Date filtering with `strdate` (newer than) and `enddate` (older than).
- Sorting with `sort` and `order`.

## Practical use in Op1 (non-invasive):

- Fetch a small, recent set of documents for `resolved_country` (and optionally sector keywords), store as metadata, and optionally extract lightweight text features (counts, topic tags, docty distribution) to feed the *linked/macro context* layer.
- Keep this as an additive source. It must not change the app's core quantitative cache logic.

**Note:** This older FRED-mapping workflow is deprecated.

Operator 1's macro layer should be **World Bank Open Data-first** (global coverage), using a config-driven indicator map (see the World Bank macro section below). If you later add a secondary provider for higher-frequency rates, keep the *canonical variable names* the same and treat World Bank as the fallback.

---

## D.2) Gemini Workflow for Linked Entity Discovery (Relationships → Eulerpool)

After the target company is confirmed, the system must discover linked entities (competitors, suppliers, customers, etc.) and resolve them to Eulerpool.

### **Gemini's role (Phase 1: Relationship Discovery):**

Gemini receives company information and must generate a list of related entities:

#### **Gemini prompt template for relationship discovery:**

You are a financial relationship mapping assistant. You need to identify companies related to a target company.

Target company:

- Name: "{company\_name}"
- Country: "{country}"
- Sector: "{sector}"
- Industry: "{industry}"
- Exchange: "{exchange}"

Your task: Identify related companies across these relationship types:

1. Competitors - Direct competitors in the same industry
2. Suppliers - Key suppliers providing inputs
3. Customers - Major customers purchasing products/services
4. Financial Institutions - Banks, insurers, investors with significant exposure
5. Logistics - Shipping, distribution, warehousing partners
6. Regulators - Government bodies, regulatory agencies

**IMPORTANT RULES:**

1. Do NOT assume specific companies - focus on relationship types and patterns
2. For each relationship type, describe WHO would typically be related and WHY
3. Suggest search terms for finding these entities in Euler pool
4. Be globally aware - relationships may cross borders
5. If relationship type is not applicable, return empty list

## 6. Output ONLY valid JSON, no additional text

Output JSON schema:

```
{  
  "relationships": {  
    "competitors": [  
      {  
        "search_term": "company name or description",  
        "country_hint": "country code or GLOBAL",  
        "sector_hint": "expected sector",  
        "reasoning": "why this is a competitor"  
      }  
    ],  
    "suppliers": [...],  
    "customers": [...],  
    "financial_institutions": [...],  
    "logistics": [...],  
    "regulators": [...]  
},  
  "relationship_count": {  
    "competitors": N,  
    "suppliers": N,  
    "customers": N,  

```

Generate the relationship map now. For each category, provide 3-7 entities if applicable, or empty list if not applicable.

### Code execution after Gemini relationship discovery:

1. Parse Gemini's JSON output
2. For each relationship category (competitors, suppliers, etc.):

- For each suggested entity:
  - a. Use entity's `search_term` to query Eulerpool search API
  - b. Apply filters: `country_hint`, `sector_hint` to narrow results
  - c. **Automatic matching algorithm** (no user confirmation for linked entities):

```

For each Eulerpool search result:
score = 0
if country matches country_hint exactly: score +
= 40
if sector matches sector_hint: score += 30
if name similarity to search_term > 0.7: score +
= 30

Select result with highest score if score >= 70 (threshold)
If no result scores >= 70, skip this entity (not found)

```

- d. Store matched entity with metadata:

```
{
  "name": "Official Eulerpool name",
  "isin": "ISIN code",
  "ticker": "Ticker",
  "country": "Country",
  "sector": "Sector",
  "relationship_type": "competitor/supplier/customer/etc",
  "match_confidence": score / 100,
  "original_search_term": "What Gemini suggested"
}
```

- 3. Build relationship graph:

```
{
  "target_company_isin": "...",
```

```

    "relationships": {
        "competitors": [
            {"isin": "...", "name": "...", "confidence": 0.9
        5},
        ...
        ],
        "suppliers": [...],
        "customers": [...],
        "financial_institutions": [...],
        "logistics": [...],
        "regulators": [...]
    },
    "total_linked_entities": N,
    "timestamp": "ISO timestamp"
}

```

#### 4. Log statistics:

- How many entities were searched for each category
- How many were successfully matched to Eulerpool
- Average match confidence per category

#### **Fallback mechanism (if Gemini provides no/few relationships):**

1. Use Eulerpool's own "similar companies" or "industry peers" endpoint if available
2. Filter companies by same sector + industry from Eulerpool search
3. Rank by market cap or relevance
4. Take top N as "competitors" category

#### **Important: No hardcoded company lists**

The system must work for ANY company globally:

- A small Malaysian semiconductor manufacturer
- A Brazilian agricultural company
- A German automotive supplier
- An Egyptian telecom operator

- A Japanese robotics firm

Gemini's relationship discovery must be based on:

- Industry knowledge patterns (not specific company names)
  - Sector relationship logic (e.g., "automotive manufacturers need steel suppliers")
  - Geographic proximity logic (e.g., "logistics partners usually in same region")
  - Market structure understanding (e.g., "oligopoly industries have few competitors")
- 

### D.3) Complete Data Source Flow Summary

1. **User Input** → Gemini generates search plan → **Eulerpool search** → User confirms → Target company resolved
  2. **Target company** → Gemini generates FRED mapping → **FRED API** → Country macro variables loaded
  3. **Target company** → Gemini discovers relationships → For each relationship:
    - Gemini search term → **Eulerpool search** → Automatic matching → Linked entity resolved
  4. **Target + all linked entities** → Fetch decision variables from Eulerpool
  5. **FRED variables + decision variables** → Compute survival modes
  6. Continue to temporal analysis...
- 

## E) Automated Discovery Workflow Guarantees

The system must guarantee:

1. **No hardcoded company data:** No ticker lists, no company name databases, no pre-built relationship maps
2. **No hardcoded FRED series:** FRED mappings are generated dynamically per country by Gemini
3. **No hardcoded sector/industry taxonomies:** Use whatever classification Eulerpool provides

4. **Fully automatic linked entity resolution:** No user confirmation for suppliers/competitors (only for target company)
5. **Graceful degradation:** If Gemini fails to find relationships, continue with sector peers only
6. **Global compatibility:** Must work for companies in any country with any naming convention

### **Validation checkpoints:**

Before proceeding to cache building, the code must validate:

- Target company resolved to valid ISIN
- Target country resolved to ISO-2 code
- World Bank macro indicators fetched (best-effort, may be low-frequency)
- At least 1 linked entity resolved (competitors or sector peers)
- If no linked entities found, use sector/industry filter on Eulerpool to find peers
- If key World Bank indicators are unavailable for the country, continue without macro (set variables to null + missing flags)

The developer must implement defensive programming:

- Try/except blocks around all Gemini calls
- Try/except blocks around all Eulerpool searches
- Try/except blocks around all FRED fetches
- Always log what succeeded vs what failed
- Never abort pipeline due to partial failures

## **E) Temporal Analysis: Complete Workflow & Mathematical Modules**

### **E.1) Overview: Day-by-Day Temporal Analysis**

Temporal analysis is the **core prediction engine** of Operator 1. It takes the 2-year cache and runs a **forward simulation** starting from the first day, predicting

each subsequent day, and comparing predictions against actual cached values. This process learns patterns, relationships, and dynamics.

### The Flow:

1. **Start:** Day 1 of cache (2 years ago)
2. **Predict:** Use current state to predict Day 2
3. **Compare:** Measure prediction error against actual Day 2 from cache
4. **Learn:** Update model parameters based on error
5. **Advance:** Move to Day 2, repeat steps 2-5
6. **Continue:** Iterate through all ~500 trading days to present
7. **Burn-Out:** Apply learned patterns repeatedly on last 6 months for accuracy improvement
8. **Predict Future:** Use finalized patterns to predict next day/week/month/year

## E.2) Mathematical Modules Suite (Comprehensive)

### Module Category 1: Regime Detection & Structural Breaks

These modules identify when the market/company enters a new "state" or experiences a fundamental shift.

#### 1. Hidden Markov Model (HMM)[1]

- Purpose: Detect hidden market regimes (bull, bear, high-vol, low-vol)
- How it works: Assumes observable variables (returns, volatility) are generated by hidden states
- Output: Regime probability per day (e.g., 70% bull regime, 30% bear regime)
- Integration: Regime probabilities become linked variables for prediction

#### 2. Gaussian Mixture Model (GMM)[1]

- Purpose: Unsupervised regime clustering from return distributions
- How it works: Fits multiple Gaussian distributions to capture different market states
- Output: Regime labels + regime-specific volatility/return characteristics

- Integration: More data-driven than HMM, no predefined regime assumptions

### **3. Bayesian Change Point Detection (BCP)[2]**

- Purpose: Identify exact days when structural breaks occur
- How it works: Bayesian inference to detect probability of regime shift at each timestamp
- Output: Change point probabilities + breakpoint locations
- Integration: Triggers model re-calibration when high-confidence break detected

### **4. PELT (Pruned Exact Linear Time)[3]**

- Purpose: Fast structural break detection for long time series
- How it works: Optimal segmentation with pruning for computational efficiency
- Output: Breakpoint locations with statistical confidence
- Integration: Pre-processing step to segment cache into stable periods

## **Module Category 2: Time Series Forecasting Core**

### **1. Adaptive Kalman Filter**

- Purpose: Optimal state estimation with noise filtering
- How it works: Recursively estimates company state variables from noisy observations
- Output: Filtered estimates of decision variables + prediction uncertainty
- Integration: Primary prediction engine for Tier 1-2 variables (liquidity, solvency)

### **2. Particle Filter (Sequential Monte Carlo)[4]**

- Purpose: Non-linear, non-Gaussian state estimation
- How it works: Uses swarm of particles to represent state distribution
- Output: Full probability distribution of future states (not just mean)
- Integration: Handles extreme events better than Kalman; use for survival mode prediction

### **3. GARCH / EGARCH (Volatility Modeling)[5]**

- Purpose: Model time-varying volatility (volatility clustering)
- How it works: Volatility depends on past volatility + past shocks
- Output: Daily volatility forecasts (Tier 3 market stability)
- Integration: Feeds into Gharar filter + survival mode detection

#### **4. VAR (Vector Autoregression)[6]**

- Purpose: Model interdependencies among multiple decision variables
- How it works: Each variable is predicted by lagged values of all variables
- Output: Joint predictions for all Tier 1-5 variables simultaneously
- Integration: Captures cross-variable dynamics (e.g., debt affects cash flow)

#### **5. LSTM (Long Short-Term Memory Networks)[7]**

- Purpose: Deep learning for complex non-linear patterns
- How it works: Neural network with memory cells to capture long-term dependencies
- Output: Pattern-based predictions for all variables
- Integration: Ensemble with traditional methods; excels at capturing regime shifts

#### **6. Transformer Architecture[7]**

- Purpose: Attention-based modeling of variable relationships
- How it works: Self-attention mechanism identifies which past days/variables matter most
- Output: Variable predictions + attention weights (interpretable feature importance)
- Integration: Modern alternative to LSTM; better at long sequences

### **Module Category 3: Causality & Relationship Analysis**

#### **1. Granger Causality Test[8]**

- Purpose: Identify which linked variables actually predict company variables
- How it works: Tests if past values of X improve prediction of Y

- Output: Causality matrix (which relationships are predictive)
- Integration: Prunes weak relationships from model; reduces overfitting

## 2. Transfer Entropy / Information Flow[9]

- Purpose: Measure directional information flow between variables
- How it works: Information-theoretic measure of predictive power
- Output: Information flow network showing variable dependencies
- Integration: Builds dynamic graph of decision + linked variable relationships

## 3. Copula Models

- Purpose: Model complex dependency structures (non-linear correlations)
- How it works: Separates marginal distributions from dependence structure
- Output: Joint distribution of variables with tail dependencies
- Integration: Captures crisis co-movements (variables crash together)

# Module Category 4: Ensemble & Optimization

## 1. Random Forest / XGBoost[10]

- Purpose: Non-parametric prediction + feature importance ranking
- How it works: Ensemble of decision trees with boosting
- Output: Predictions + variable importance scores (which Tiers matter most)
- Integration: Validates hierarchy weights; identifies overlooked relationships

## 2. Genetic Algorithm for Meta-Optimization

- Purpose: Optimize module hyperparameters and ensemble weights
- How it works: Evolutionary search through parameter space
- Output: Optimal configuration for each module (e.g., HMM states, Kalman noise)
- Integration: Meta-layer that tunes all other modules

### **3. Sobol Global Sensitivity Analysis**

- Purpose: Decompose prediction variance to identify key drivers
- How it works: Variance-based sensitivity indices for each input variable
- Output: Quantifies how much each decision/linked variable affects predictions
- Integration: Validates survival hierarchy; identifies surprising dependencies

## **Module Category 5: Monte Carlo Simulation Suite**

### **1. Regime-Aware Monte Carlo[11]**

- Purpose: Generate probabilistic scenarios conditional on regime
- How it works: Sample from regime-specific distributions (learned from history)
- Output: 10,000 possible future paths for each variable
- Integration: Provides uncertainty bands around point predictions

### **2. Importance Sampling Monte Carlo**

- Purpose: Focus simulation on rare but critical events (survival scenarios)
- How it works: Over-sample tail events, re-weight for unbiased estimates
- Output: Accurate tail risk estimates ( $P(\text{survival failure})$ )
- Integration: Critical for vanity-adjusted survival probability

## **Module Category 6: Pattern Recognition for Charts**

### **1. Candlestick Pattern Detector**

- Purpose: Identify traditional technical patterns (doji, hammer, engulfing, etc.)
- How it works: Rule-based + ML hybrid to detect OHLC patterns
- Output: Pattern labels + formation probabilities for next day/week/month
- Integration: Generates predicted candlestick series for report

### **2. Wave/Cycle Decomposition (Fourier/Wavelet)**

- Purpose: Extract cyclical components (earnings cycles, seasonal patterns)
- How it works: Frequency domain analysis or multi-resolution wavelets
- Output: Dominant cycles + phase information
- Integration: Improves prediction of periodic events (quarterly earnings)

### **E.3) Module Integration & Workflow Design**

#### **How Modules Work Together (Synergies):**

##### **Synergy 1: Regime Detection → Regime-Conditional Prediction**

- HMM/GMM detects current regime
- GARCH, Kalman, VAR each train separate models per regime
- Prediction switches between regime-specific models based on HMM output
- Result: Better accuracy because model adapts to market state

##### **Synergy 2: Structural Break Detection → Model Reset**

- PELT/BCP run continuously to detect breaks
- When break detected, trigger:
  - Kalman filter reset (re-initialize state)
  - LSTM/Transformer retraining on post-break data only
  - Survival hierarchy recalibration
- Result: Prevents outdated patterns from degrading predictions

##### **Synergy 3: Causality Pruning → Feature Selection**

- Granger causality + Transfer Entropy identify weak linked variables
- Prune non-predictive relationships before training VAR/LSTM
- Random Forest importance validates pruning decisions
- Result: Simpler, faster, less overfit models

##### **Synergy 4: Ensemble Fusion**

- Each module generates predictions independently:
  - Kalman: linear optimal estimate

- LSTM: non-linear pattern capture
- VAR: multi-variable dynamics
- Random Forest: non-parametric robustness
- Genetic algorithm learns optimal weighted combination
- Weights vary by hierarchy tier (Kalman dominates Tier 1, LSTM dominates Tier 5)
- Result: Best of all approaches

### **Synergy 5: Monte Carlo Uncertainty Quantification**

- Point predictions from ensemble
- Regime-aware Monte Carlo generates distribution around point
- Copula models tail dependencies for extreme scenarios
- Importance sampling over-samples survival-critical paths
- Result: Full probability distribution, not just single prediction

### **Synergy 6: Sensitivity Analysis → Hierarchy Validation**

- Sobol decomposes variance in predictions
- Compare Sobol indices to survival hierarchy weights
- If mismatch (e.g., Tier 3 variable has higher Sobol than Tier 2), flag for review
- Result: Data-driven validation of expert-designed hierarchy

## **E.4) Temporal Analysis Execution Sequence**

### **Phase 1: Setup (Day 0)**

1. Load 2-year cache (all decision + linked variables)
2. Detect historical regimes using HMM/GMM on full cache
3. Identify structural breaks using PELT/BCP
4. Segment cache into stable periods between breaks
5. Compute Granger causality matrix to prune weak relationships
6. Initialize all module parameters (Kalman noise, LSTM architecture, etc.)

## Phase 2: Forward Pass Training (Day 1 → Day ~500)

For each day  $t$  from start to present:

### Step A: Regime Classification

- HMM: What regime are we in today?
- Output: `regime_t` (e.g., "bull", "bear", "high-vol", "crisis")

### Step B: Multi-Module Prediction

- Kalman Filter: Predict Tier 1-2 variables (liquidity, solvency)
- VAR: Predict all tiers jointly using lagged values
- GARCH: Predict Tier 3 volatility
- LSTM/Transformer: Predict all tiers using learned patterns
- Random Forest: Predict using non-linear tree ensemble

### Step C: Ensemble Fusion

- Weight each module's prediction based on:
  - Historical accuracy
  - Current regime (LSTM weighted higher in volatile regimes)
  - Hierarchy tier (Kalman higher weight for Tier 1)
- Produce final prediction: `prediction_t+1`

### Step D: Reality Check

- Retrieve actual values from cache: `actual_t+1`
- Compute errors: `error_t+1 = actual_t+1 - prediction_t+1`
- Weighted error using hierarchy weights (Tier 1 errors count more)

### Step E: Online Learning

- Update Kalman gain
- Update LSTM weights via backpropagation
- Update ensemble weights via gradient descent
- If BCP detects structural break, reset models

### Step F: Advance

- Set `t = t + 1`

- Repeat steps A-E

### **Phase 3: Burn-Out Process (Last ~130 Days)**

Burn-out is **intensive re-training on recent data** to maximize near-term accuracy.

#### **Iteration 1:**

1. Take last 6 months of cache (~130 days)
2. Run forward pass (steps A-F) but with higher learning rates
3. Measure final prediction accuracy on most recent 20 days
4. Store learned patterns as `pattern_v1`

#### **Iteration 2-10:**

1. Reset to 6 months ago
2. Initialize with `pattern_v{i-1}`
3. Run forward pass with:
  - Regime-specific models (separate parameters per regime)
  - Increased emphasis on survival mode days (higher error weight)
  - Vanity-adjusted hierarchy weights
4. Measure accuracy on last 20 days
5. If accuracy improved, store as `pattern_v{i}`
6. Repeat

#### **Convergence:**

- Stop when accuracy stops improving (typically 5-10 iterations)
- Final output: `pattern_final` with optimized parameters for each module

### **Phase 4: Future Prediction (Present → Future)**

Now predict variables for next day/week/month/year.

#### **For Next Day (Day +1):**

1. Current state = Last day of cache (today)
2. HMM: Classify current regime
3. All modules predict using `pattern_final`

4. Ensemble fusion with hierarchy weights
5. Monte Carlo: Generate 10,000 scenarios around point prediction
6. Output:
  - Point predictions for all Tier 1-5 variables
  - Prediction intervals (5%, 50%, 95%)
  - Predicted OHLC (but mask High/Close/Open per Technical Alpha protection)

**For Next Week (Days +1 to +5):**

1. Predict Day +1 as above
2. Use Day +1 prediction as input to predict Day +2
3. Repeat iteratively for 5 days
4. Output: Full candlestick series (no masking for week)

**For Next Month (Days +1 to +21):**

- Same iterative process, 21 steps forward
- Output: Full monthly chart projection

**For Next Year (Days +1 to +252):**

- Same iterative process, 252 steps forward
- Include uncertainty bands (widen over time)
- Output: Annual chart with confidence intervals

## E.5) From Burn-Out Results to Complete Company Profile

**The burn-out process produces:**

1. **Learned Patterns ( `pattern_final` )**
  - Optimized parameters for all modules
  - Regime-specific models
  - Ensemble weights
  - Hierarchy weights validation

2. **Historical Performance Metrics**

- Prediction accuracy per tier
- Regime classification accuracy
- Structural break detection success rate
- Module contribution scores

### 3. Future Predictions

- Next day: all variables + masked OHLC (Low only)
- Next week: all variables + full OHLC
- Next month: all variables + full OHLC
- Next year: all variables + full OHLC with uncertainty

#### Building the Complete Company Profile:

The profile is a **comprehensive JSON/dict object** containing:

```
company_profile = {
    # Identity
    "company": {"name": ..., "ticker": ..., "isin": ..., "sector": ..., "industry": ...},

    # Current State (Latest Day)
    "current_state": {
        "date": "2026-02-09",
        "regime": "Emerging Expansion",
        "survival_flags": {
            "company_survival_mode": False,
            "country_survival_mode": False,
            "country_protected": True,
        },
        "vanity_percentage": 3.2,
        "hierarchy_regime": "Normal",
        "tier1_liquidity": {"cash_ratio": 0.45, "fcf_ttm": 12.5e9, ...},
        "tier2_solvency": {"debt_to_equity": 0.85, "interest_coverage": 8.2, ...},
        "tier3_market": {"volatility_21d": 0.18, "drawdown_252d": -0.12, ...},
    }
}
```

```

        "tier4_profitability": {"gross_margin": 0.42, "operating_margin": 0.18, ...},
        "tier5_valuation": {"pe_ratio": 24.5, "ev_to_ebitda": 14.2, ...},
    },

    # Historical Performance (2 Years)
    "historical": {
        "date_range": ["2024-02-09", "2026-02-09"],
        "return_total": 0.34, # 34% total return
        "return_real": 0.28, # 28% after inflation (Purchasing Power filter)
        "volatility_annualized": 0.22,
        "sharpe_ratio": 1.4,
        "max_drawdown": -0.18,
        "survival_days": 0,      # Days in survival mode
        "regime_breakdown": {"bull": 320, "bear": 45, "high_vol": 135},
    },

    # Linked Variables (Current)
    "linked_variables": {
        "sector_performance": {"median_return_21d": 0.05, "rel_strength": +0.02},
        "competitor_health": {"avg_debt_to_equity": 1.2, "relative_position": "strong"},
        "macro_environment": {
            "inflation_rate": 2.8,
            "policy_rate": 4.5,
            "yield_curve_slope": 0.6,
            "credit_spread": 1.2,
        },
    },
}

# Predictions (Next Day)
"predictions_next_day": {
    "date": "2026-02-10",
    "ohlc": {"low": 152.3}, # Only Low shown (Technica

```

```

    "tier1_liquidity": {"cash_ratio": [0.44, 0.45, 0.46]}, # [5%, 50%, 95%]
    "tier2_solvency": {"debt_to_equity": [0.83, 0.85, 0.88]},
    # ... all other tiers
    "survival_probability": 0.998, # 99.8% probability of not entering survival
  },

  # Predictions (Next Week)
  "predictions_next_week": {
    "date_range": ["2026-02-10", "2026-02-14"],
    "ohlc_series": [...], # Full OHLC candlesticks
    "predicted_return": 0.02, # +2% expected
    "volatility_forecast": 0.19,
  },

  # Predictions (Next Month)
  "predictions_next_month": {
    "date_range": ["2026-02-10", "2026-03-10"],
    "ohlc_series": [...],
    "predicted_return": 0.06, # +6% expected
    "key_events": ["Earnings on 2026-02-28"],
  },

  # Predictions (Next Year)
  "predictions_next_year": {
    "date_range": ["2026-02-10", "2027-02-09"],
    "ohlc_series_monthly": [...], # Monthly aggregates to avoid clutter
    "predicted_return": [0.08, 0.15, 0.23], # [5%, 50%, 95%]
    "predicted_regime_shifts": [{"date": "2026-Q3", "to_regime": "high_vol"}],
  },

  # Model Performance & Confidence

```

```

"model_metrics": {
    "burn_out_iterations": 7,
    "final_accuracy_tier1": 0.94, # 94% accurate on Tier 1
    "final_accuracy_tier2": 0.89,
    "final_accuracy_tier3": 0.76,
    "final_accuracy_tier4": 0.71,
    "final_accuracy_tier5": 0.63,
    "regime_classification_accuracy": 0.88,
    "best_performing_module": "Ensemble (Kalman+LSTM)",
},
}

# Ethical Filters (Applied)
"filters": {
    "purchasing_power": {"nominal_return": 0.34, "real_return": 0.28, "verdict": "PASS"},
    "solvency": {"debt_to_equity": 0.85, "threshold": 3.0, "verdict": "PASS - Stable"},
    "gharar": {"volatility_21d": 0.18, "stability_score": 7.8, "verdict": "MODERATE - Calculated Risk"},
    "cash_is_king": {"fcf_yield": 0.048, "verdict": "PASS - Healthy Cash Generation"},
},
}

# Vanity Analysis
"vanity_analysis": {
    "vanity_percentage": 3.2,
    "interpretation": "Normal, acceptable level",
    "components": {
        "exec_comp_excess": 0.8e9,
        "sga_bloat": 0.2e9,
        "vanity_buybacks": 0.0,
        "marketing_excess": 0.0,
    },
},
}

# Candlestick Patterns Detected
"technical_patterns": {
}

```

```
        "recent_patterns": ["Bullish Engulfing (2026-02-05)", "Morning Star (2026-01-28)"],  
        "predicted_patterns_week": ["Doji (2026-02-11)", "Hammer (2026-02-13)"],  
    },  
}
```

This complete profile is then passed to Gemini API with the prompt:

You are a Bloomberg-style financial analyst. Using the comprehensive company profile below, generate a professional investment report that includes:

1. Executive Summary (key findings in 3 bullet points)
2. Company Overview (current state, sector position)
3. Historical Performance Analysis (2-year track record)
4. Current Financial Health (Tier 1-5 breakdown with hierarchy explanation)
5. Linked Variables & Market Context (sector, competitors, macro)
6. Predictions & Forecasts (day/week/month/year with uncertainty)
7. Risk Assessment (survival probability, ethical filters, vanity analysis)
8. Technical Patterns & Charts (candlestick predictions)
9. Investment Recommendation (Buy/Hold/Sell with rationale)
10. Appendix (model methodology, confidence metrics)

Profile data:

{company\_profile}

Format as a professional PDF-ready report with sections, tables, and chart descriptions.

Gemini generates the final Bloomberg-style report, which is then exported as PDF.

## F) Complete Company Profile Variables (Post Burn-Out)

After the burn-out process completes, the system must build a **comprehensive company profile** that contains all variables needed for the final Bloomberg-style report. This profile is a structured data object (JSON/dict) that Gemini will consume.

## F.1) Variable Categories in the Complete Profile

The complete profile organizes variables into **9 major categories**:

### Category 1: Company Identity & Classification

These are static or slow-changing variables that identify the company:

- `company_name` - Full legal name
- `ticker` - Primary ticker symbol
- `isin` - International Securities Identification Number
- `exchange` - Primary exchange (e.g., NYSE, NASDAQ)
- `country` - Country of incorporation
- `sector` - GICS sector classification
- `industry` - GICS industry classification
- `sub_industry` - GICS sub-industry (if available)
- `currency` - Reporting currency
- `market_cap_current` - Current market capitalization
- `shares_outstanding_current` - Current shares outstanding

**How to get them:** Direct extraction from Eulerpool profile endpoint, stored in cache metadata.

### Category 2: Current State Snapshot (Latest Day)

These are the most recent values (as of cache end date) for all Tier 1-5 variables:

#### Tier 1 - Liquidity & Cash:

- `cash_and_equivalents_asof` - Cash on hand (millions/billions in reporting currency)
- `cash_ratio` - Cash / Current Liabilities
- `free_cash_flow_ttm` - TTM free cash flow

- `operating_cash_flow_asof` - Latest operating cash flow
- `current_ratio` - Current assets / Current liabilities

### Tier 2 - Solvency & Debt:

- `total_debt_asof` - Total debt (short-term + long-term)
- `debt_to_equity` - Total debt / Total equity
- `net_debt` - Total debt - Cash
- `net_debt_to_ebitda` - Net debt / EBITDA TTM
- `interest_coverage` - EBIT / Interest expense
- `total_equity_asof` - Total shareholder equity

### Tier 3 - Market Stability:

- `volatility_21d` - 21-day rolling volatility (annualized)
- `drawdown_252d` - Maximum drawdown over 252 days
- `volume_avg_21d` - 21-day average volume
- `beta_252d` - 252-day beta vs market index (if available)
- `close_current` - Current stock price

### Tier 4 - Profitability:

- `gross_margin` - Gross profit / Revenue
- `operating_margin` - Operating income / Revenue
- `net_margin` - Net income / Revenue
- `roe` - Return on equity (Net income / Avg equity)
- `roa` - Return on assets (Net income / Total assets)
- `revenue_ttm_asof` - TTM revenue
- `net_income_ttm_asof` - TTM net income
- `ebitda_ttm_asof` - TTM EBITDA

### Tier 5 - Growth & Valuation:

- `revenue_growth_yoy` - Year-over-year revenue growth rate
- `earnings_growth_yoy` - Year-over-year earnings growth rate
- `pe_ratio` - Price / Earnings (TTM)

- `earnings_yield` - Earnings / Price (inverse of P/E)
- `ps_ratio` - Price / Sales
- `pb_ratio` - Price / Book value
- `ev_to_ebitda` - Enterprise value / EBITDA
- `enterprise_value` - Market cap + Net debt

**How to get them:** Extract from the **last row** of the target company's 2-year daily cache after all features have been computed.

### Category 3: Historical Performance Metrics (2-Year Summary)

These summarize the company's performance over the full 2-year cache period:

- `date_range_start` - First date in cache
- `date_range_end` - Last date in cache (present)
- `return_total` - Total return over 2 years:  $(\text{close\_end} / \text{close\_start}) - 1$
- `return_annualized` - Annualized return:  $((1 + \text{return\_total}) ^ (1/2)) - 1$
- `return_real` - Total return adjusted for inflation (Purchasing Power filter)
- `volatility_annualized` - Annualized volatility over 2 years
- `sharpe_ratio` -  $(\text{Annualized return} - \text{risk\_free\_rate}) / \text{Volatility}$
- `max_drawdown_historical` - Worst peak-to-trough decline in 2 years
- `recovery_time_avg` - Average days to recover from drawdowns
- `up_days_percentage` - Percentage of days with positive returns
- `down_days_percentage` - Percentage of days with negative returns
- `best_day_return` - Largest single-day gain
- `worst_day_return` - Largest single-day loss

**How to get them:** Compute from the full 2-year cache using pandas aggregations on the return and volatility series.

### Category 4: Survival Mode Analysis (Historical + Current)

These track survival mode states over time and current status:

#### Historical Survival Metrics:

- `survival_days_company` - Total days company was in survival mode (2 years)

- `survival_days_country` - Total days country was in survival mode (2 years)
- `survival_days_both` - Days when both were in survival mode
- `survival_episodes_count` - Number of distinct survival mode episodes
- `longest_survival_episode` - Longest consecutive days in survival mode
- `avg_survival_episode_length` - Average length of survival episodes

### Current Survival Status:

- `company_survival_mode_flag` - Boolean: Is company currently in survival mode?
- `country_survival_mode_flag` - Boolean: Is country currently in survival mode?
- `country_protected_flag` - Boolean: Is company protected by government?
- `survival_regime_current` - String label ("Normal", "Company Survival", "Country Crisis", "Extreme Survival")

### Hierarchy Weights (Current):

- `hierarchy_tier1_weight` - Current Tier 1 weight (0-100)
- `hierarchy_tier2_weight` - Current Tier 2 weight (0-100)
- `hierarchy_tier3_weight` - Current Tier 3 weight (0-100)
- `hierarchy_tier4_weight` - Current Tier 4 weight (0-100)
- `hierarchy_tier5_weight` - Current Tier 5 weight (0-100)

### Vanity Analysis:

- `vanity_percentage_current` - Current vanity spending as % of revenue
- `vanity_percentage_avg_2y` - Average vanity % over 2 years
- `vanity_interpretation` - String label ("Disciplined", "Normal", "Elevated", "High", "Extreme")
- `exec_comp_excess` - Excess executive compensation (absolute value)
- `sga_bloat` - SG&A expenses above industry median (absolute value)
- `vanity_buybacks` - Stock buybacks while FCF negative (absolute value)
- `marketing_excess` - Marketing expenses above threshold during distress

**How to get them:** Compute from the survival mode flags and hierarchy weight columns in the cache. Use pandas groupby and aggregations to count survival days and episodes.

## Category 5: Regime Classification (Historical Distribution)

These show how much time the company spent in each market regime:

- `regime_breakdown` - Dict mapping regime names to day counts:
  - `bull` - Days in bull market regime
  - `bear` - Days in bear market regime
  - `high_vol` - Days in high volatility regime
  - `low_vol` - Days in low volatility regime
  - `crisis` - Days in crisis regime (if detected)
- `regime_current` - Current regime label (string)
- `regime_transitions_count` - Number of times regime changed in 2 years
- `structural_breaks_detected` - List of dates where structural breaks occurred
- `structural_breaks_count` - Total number of structural breaks detected

**How to get them:** Extract from the regime classification columns added by HMM/GMM modules during temporal analysis. Count occurrences of each regime label.

## Category 6: Linked Variables (Current External Context)

These describe the company's current position relative to its environment:

### Sector Performance:

- `sector_median_return_21d` - Median 21-day return of sector peers
- `sector_median_volatility_21d` - Median volatility of sector
- `rel_strength_vs_sector` - Company return - Sector median return
- `sector_rank_percentile` - Percentile rank within sector (0-100)

### Industry Performance:

- `industry_median_return_21d` - Median 21-day return of industry peers
- `industry_median_debt_to_equity` - Median leverage of industry
- `industry_median_fcf_yield` - Median FCF yield of industry
- `valuation_premium_vs_industry` - Company P/E - Industry median P/E

### Competitor Health:

- `competitors_count` - Number of identified competitors
- `competitors_avg_return_21d` - Average competitor return
- `competitors_avg_volatility_21d` - Average competitor volatility
- `competitors_avg_debt_to_equity` - Average competitor leverage
- `relative_position_vs_competitors` - String label ("Leading", "Strong", "Average", "Weak", "Lagging")

### Supply Chain Aggregates:

- `suppliers_count` - Number of identified suppliers
- `customers_count` - Number of identified customers
- `supply_chain_avg_health_score` - Average health score of supply chain entities
- `supply_chain_stress_flag` - Boolean: Is supply chain under stress?

### Macro Environment (from FRED):

- `inflation_rate` - Current inflation rate (annualized %)
- `policy_rate` - Current central bank policy rate
- `yield_curve_slope` - Long rate - Short rate (basis points)
- `credit_spread` - Corporate credit spread (basis points)
- `unemployment_rate` - Current unemployment rate
- `gdp_growth_latest` - Most recent GDP growth rate (quarterly annualized)
- `fx_volatility` - Currency volatility vs USD

**How to get them:** Extract from linked aggregates cache and FRED cache. For current values, use the last row. For relative metrics, compute company value minus sector/industry median.

### Category 7: Model Performance & Confidence Metrics

These quantify how well the temporal models performed and their reliability:

- `burn_out_iterations` - Number of burn-out iterations completed
- `burn_out_converged` - Boolean: Did burn-out converge early?
- `final_accuracy_tier1` - Prediction accuracy for Tier 1 variables (0-1)
- `final_accuracy_tier2` - Prediction accuracy for Tier 2 variables (0-1)

- `final_accuracy_tier3` - Prediction accuracy for Tier 3 variables (0-1)
- `final_accuracy_tier4` - Prediction accuracy for Tier 4 variables (0-1)
- `final_accuracy_tier5` - Prediction accuracy for Tier 5 variables (0-1)
- `regime_classification_accuracy` - Accuracy of HMM regime detection (0-1)
- `structural_break_precision` - Precision of breakpoint detection (0-1)
- `best_performing_module` - String name of best prediction module
- `module_contribution_scores` - Dict mapping module names to contribution %:
  - `kalman` - Kalman filter contribution
  - `var` - VAR model contribution
  - `lstm` - LSTM contribution
  - `rf` - Random forest contribution
  - `garch` - GARCH contribution
- `ensemble_weights_final` - Dict of final ensemble weights per module
- `causality_network_density` - Granger causality network density (0-1)
- `prediction_confidence_current` - Overall confidence in next-day prediction (0-1)

**How to get them:** These are outputs from the temporal analysis pipeline. Store them as metadata during burn-out and ensemble optimization.

### Category 8: Predictions (Future Forecasts)

These are the forward-looking predictions for multiple time horizons:

#### Next Day Predictions (+1 day):

- `pred_date_next_day` - Prediction target date
- `pred_ohlc_next_day` - Dict with only Low shown (Technical Alpha protection):
  - `low` - Predicted low price (single value)
  - `high` - MASKED
  - `open` - MASKED
  - `close` - MASKED
- `pred_tier1_next_day` - Dict of Tier 1 variable predictions with uncertainty:
  - Each variable as `{name: [p5, p50, p95]}` (5th, 50th, 95th percentiles)

- `pred_tier2_next_day` - Dict of Tier 2 predictions with uncertainty
- `pred_tier3_next_day` - Dict of Tier 3 predictions with uncertainty
- `pred_tier4_next_day` - Dict of Tier 4 predictions with uncertainty
- `pred_tier5_next_day` - Dict of Tier 5 predictions with uncertainty
- `pred_survival_probability_next_day` - Probability company avoids survival mode (0-1)
- `pred_regime_next_day` - Most likely regime for next day
- `pred_regime_probabilities_next_day` - Dict of regime probabilities

### **Next Week Predictions (+5 days):**

- `pred_date_range_next_week` - [start\_date, end\_date]
- `pred_ohlc_series_next_week` - Full OHLC candlestick series (list of dicts)
- `pred_return_next_week` - Expected return over next week (p50)
- `pred_volatility_next_week` - Expected volatility over next week
- `pred_patterns_next_week` - List of expected candlestick patterns with dates

### **Next Month Predictions (+21 days):**

- `pred_date_range_next_month` - [start\_date, end\_date]
- `pred_ohlc_series_next_month` - Full OHLC candlestick series (list of dicts)
- `pred_return_next_month` - Expected return over next month [p5, p50, p95]
- `pred_volatility_next_month` - Expected volatility over next month
- `pred_key_events_next_month` - List of predicted key events (e.g., earnings dates)
- `pred_regime_shifts_next_month` - List of predicted regime changes with dates

### **Next Year Predictions (+252 days):**

- `pred_date_range_next_year` - [start\_date, end\_date]
- `pred_ohlc_series_next_year` - Monthly aggregated OHLC series (to reduce clutter)
- `pred_return_next_year` - Expected return over next year [p5, p50, p95]
- `pred_volatility_next_year` - Expected volatility over next year
- `pred_regime_shifts_next_year` - List of predicted regime changes with quarters

- `pred_uncertainty_band_next_year` - Width of prediction interval (indicates confidence decay)

## Monte Carlo Scenarios:

- `monte_carlo_scenarios_count` - Number of simulations run (typically 10,000)
- `monte_carlo_percentiles` - Dict of percentile values for key variables:
  - Format: `{variable_name: [p5, p10, p25, p50, p75, p90, p95]}`
- `tail_risk_probability` - Probability of extreme negative outcome (survival failure)
- `tail_gain_probability` - Probability of extreme positive outcome

**How to get them:** These are outputs from the prediction phase after burn-out.

Each prediction module generates forecasts, ensemble combines them, and Monte Carlo adds uncertainty quantification.

## Category 9: Ethical Filter Results

These show the pass/fail status of each ethical filter:

### Purchasing Power Filter:

- `filter_purchasing_power_verdict` - String ("PASS", "FAIL", "WARNING")
- `filter_purchasing_power_nominal_return` - Nominal return over 2 years
- `filter_purchasing_power_real_return` - Real return after inflation adjustment
- `filter_purchasing_power_inflation_impact` - Difference between nominal and real

### Solvency Filter:

- `filter_solvency_verdict` - String ("PASS - Stable", "WARNING - Elevated", "FAIL - Fragile")
- `filter_solvency_debt_to_equity` - Current debt-to-equity ratio
- `filter_solvency_threshold` - Threshold used (typically 3.0)
- `filter_solvency_interpretation` - Detailed string explanation

### Gharar Filter:

- `filter_gharar_verdict` - String ("LOW - Stable", "MODERATE - Calculated Risk", "HIGH - Speculation", "EXTREME - Gambling")
- `filter_gharar_volatility` - Current 21-day volatility

- `filter_gharar_stability_score` - Stability score (0-10, higher is more stable)
- `filter_gharar_interpretation` - Detailed string explanation

### Cash is King Filter:

- `filter_cash_verdict` - String ("PASS - Strong", "PASS - Healthy", "WARNING - Weak", "FAIL - Burning Cash")
- `filter_cash_fcf_yield` - Free cash flow yield
- `filter_cash_fcf_margin` - Free cash flow margin
- `filter_cash_interpretation` - Detailed string explanation

**How to get them:** Compute these as post-processing after all features are available. Each filter applies thresholds and rules to current state variables.

## F.2) How to Extract Variables from Temporal Analysis Results

### Step 1: Extract from Cache

Most variables come directly from the final 2-year cache DataFrame:

```
# Last row = current state
current = cache.iloc[-1]

# Historical aggregations
historical_return = (cache['close'].iloc[-1] / cache['close'].iloc[0]) - 1
max_drawdown = cache['drawdown_252d'].min()
volatility_annualized = cache['volatility_21d'].mean() * np.sqrt(252)
```

### Step 2: Extract from Temporal Analysis Metadata

Model performance metrics are stored during temporal analysis:

```
# From forward pass
accuracy_tier1 = 1 - mean(errors_by_tier[1][-20:]) # Last 20 days

# From burn-out
```

```
best_patterns = burn_out_results['best_patterns']
iterations = burn_out_results['iterations_count']
```

### Step 3: Extract from Prediction Outputs

Predictions are generated after burn-out:

```
# From ensemble predictor
pred_next_day = ensemble.predict(current_state, tier=1)
percentiles = monte_carlo.get_percentiles([5, 50, 95])
```

### Step 4: Extract from Linked Caches

Linked variables come from aggregating peer/competitor caches:

```
# Sector median
sector_peers_caches = [c for c in linked_caches if c.sector
== target.sector]
sector_median_return = np.median([c['return_21d'].iloc[-1]
for c in sector_peers_caches])
```

### Step 5: Extract from FRED Cache

Macro variables come from FRED:

```
# From country-specific FRED cache
inflation_rate = fred_cache['inflation_rate'].iloc[-1]
policy_rate = fred_cache['policy_rate'].iloc[-1]
```

### Step 6: Compute Derived Profile Variables

Some profile variables require additional computation:

```
# Regime breakdown
regime_counts = cache['regime_label'].value_counts().to_dict()

# Survival episodes
survival_episodes = detect_episodes(cache['company_survival
_mode_flag'])
```

```
# Relative strength
rel_strength = current['return_21d'] - sector_median_return
```

### F.3) Complete Profile Structure Template

```
company_profile = {
    "company": {
        "name": str,
        "ticker": str,
        "isin": str,
        "exchange": str,
        "country": str,
        "sector": str,
        "industry": str,
        "currency": str,
        "market_cap": float,
    },
    "current_state": {
        "date": str,
        "regime": str,
        "tier1": {dict of Tier 1 variables},
        "tier2": {dict of Tier 2 variables},
        "tier3": {dict of Tier 3 variables},
        "tier4": {dict of Tier 4 variables},
        "tier5": {dict of Tier 5 variables},
        "survival": {dict of survival flags and weights},
        "vanity": {dict of vanity metrics},
    },
    "historical": {dict of 2-year summary metrics},
    "linked_variables": {
        "sector": {dict},
        "industry": {dict},
        "competitors": {dict},
        "supply_chain": {dict},
        "macro": {dict},
    },
    "predictions": {
```

```

    "next_day": {dict},
    "next_week": {dict},
    "next_month": {dict},
    "next_year": {dict},
    "monte_carlo": {dict},
},
"model_metrics": {dict of model performance},
"filters": {
    "purchasing_power": {dict},
    "solvency": {dict},
    "gharar": {dict},
    "cash_is_king": {dict},
},
"regime_analysis": {dict of regime breakdown},
"patterns": {dict of detected and predicted patterns},
}

```

## G) Gemini Report Generation Workflow

After the complete company profile is built, it is passed to the **Gemini API** to generate a professional Bloomberg-style investment report.

### G.1) Report Generation Philosophy

The report must:

- Be **comprehensive yet accessible** - all technical depth is present but explained clearly
- Be **visual-first** - describe charts, tables, and visualizations that convey insights at a glance
- Be **action-oriented** - conclude with clear investment recommendation and rationale
- Be **ethical-first** - prominently feature the four filters and survival mode analysis
- Be **hierarchical** - respect the Tier 1-5 hierarchy, emphasizing liquidity and solvency in survival scenarios

## G.2) Gemini Prompt Structure

The prompt to Gemini has **three parts**:

### Part 1: Role & Context

Define Gemini's role and the type of report needed:

You are a Bloomberg-style financial analyst specializing in comprehensive equity research.

You have been provided with a complete company profile that includes:

- 2 years of historical financial and market data
- Advanced temporal analysis using 20+ mathematical models
- Survival mode analysis (company + country)
- Ethical filter assessments (Islamic finance compatible)
- Multi-horizon predictions with uncertainty quantification

Your task is to generate a professional investment report that synthesizes this information into actionable insights for sophisticated investors.

### Part 2: Report Structure Requirements

Specify the exact sections Gemini must produce:

The report must include the following sections:

#### 1. EXECUTIVE SUMMARY

- 3 bullet points: key findings
- Investment recommendation: BUY / HOLD / SELL with confidence level
- Target price with time horizon

#### 2. COMPANY OVERVIEW

- Identity and classification
- Current market position
- Sector and industry context

#### 3. HISTORICAL PERFORMANCE ANALYSIS (2 Years)

- Total and real returns (Purchasing Power filter applied)
  - Risk-adjusted performance (Sharpe ratio, max drawdown)
  - Regime breakdown (time in bull/bear/high-vol)
  - Structural breaks and major events
4. CURRENT FINANCIAL HEALTH (Tier-by-Tier Breakdown)
- Tier 1: Liquidity & Cash (Cash is King filter results)
  - Tier 2: Solvency & Debt (Solvency filter results)
  - Tier 3: Market Stability (Gharar filter results)
  - Tier 4: Profitability Quality
  - Tier 5: Growth & Valuation
  - Hierarchy explanation: why tiers matter and current weights
5. SURVIVAL MODE ANALYSIS
- Current survival status (company, country, protection)
  - Historical survival episodes
  - Vanity expenditure analysis and interpretation
  - Survival probability and risk factors
6. LINKED VARIABLES & MARKET CONTEXT
- Sector performance and relative strength
  - Industry positioning and valuation premium/discount
  - Competitor health assessment
  - Supply chain risk analysis
  - Macro environment (FRED indicators)
7. TEMPORAL ANALYSIS & MODEL INSIGHTS
- Regime classification and current regime
  - Causality network highlights (key variable relationships)
  - Model performance summary
  - Confidence levels by tier
8. PREDICTIONS & FORECASTS
- Next day: Tier 1-5 predictions with uncertainty (Low only for OHLC)

- Next week: Expected return, volatility, candlestick patterns
- Next month: Price target range, key events
- Next year: Annual outlook with regime shift predictions
- Monte Carlo tail risk and upside scenarios

## 9. TECHNICAL PATTERNS & CHART ANALYSIS

- Historical patterns detected (last 6 months)
- Predicted patterns (next week/month)
- Chart interpretation and technical signals

## 10. ETHICAL FILTER ASSESSMENT

- Purchasing Power: Real vs nominal return verdict
- Solvency: Leverage fragility assessment
- Gharar: Speculation vs calculated risk verdict
- Cash is King: Cash generation quality verdict
- Overall ethical score and interpretation

## 11. RISK FACTORS & LIMITATIONS

- Model assumptions and limitations
- Key risks to forecast (company-specific and macro)
- Scenarios that could invalidate predictions

## 12. INVESTMENT RECOMMENDATION

- Clear BUY / HOLD / SELL recommendation
- Target price with 12-month horizon
- Confidence level (High / Medium / Low)
- Key catalysts to watch
- Entry and exit strategy

## 13. APPENDIX

- Methodology summary (temporal modules used)
- Variable tier definitions
- Glossary of technical terms
- Data sources and timestamps

## Part 3: Profile Data Injection

Inject the complete company profile as JSON:

```
Complete Company Profile:  
{json.dumps(company_profile, indent=2, default=str)}
```

Generate the report now.

### G.3) Gemini Output Processing

Gemini returns markdown-formatted text. The system must:

#### Step 1: Validate Output

- Check that all required sections are present
- Verify investment recommendation is clear (BUY/HOLD/SELL)
- Ensure no hallucinated data (cross-check numbers against profile)

#### Step 2: Enhance with Charts

Since Gemini generates text descriptions of charts, the system should:

- Generate actual chart images using matplotlib/plotly:
  - Price history chart with regime shading
  - Candlestick chart for next week/month predictions
  - Tier hierarchy visual (bar chart of weights)
  - Survival mode timeline
  - Monte Carlo distribution plots
- Insert chart image references into markdown

#### Step 3: Format for PDF

Convert enhanced markdown to PDF using pandoc or similar:

```
pandoc investment_report.md -o investment_report.pdf --pdf-engine=xelatex
```

### G.4) Report Presentation: Modified Bloomberg Style

#### Visual Design Principles:

## **1. First Page: Executive Dashboard**

- Company logo and key metrics in a grid
- Large investment recommendation badge (BUY/HOLD/SELL)
- Snapshot chart (2-year price + regime shading)
- Tier hierarchy bar chart
- Ethical filter traffic lights (green/yellow/red)

## **2. Section Headers with Icons**

- Each major section gets a visual icon
- Color coding: Blue (data), Green (positive), Red (risk), Gold (prediction)

## **3. Data Tables: Tier-Organized**

- All financial data presented in Tier 1-5 tables
- Current values + Historical averages + Predictions
- Color-coded cells (green = strong, yellow = neutral, red = weak)

## **4. Charts: Regime-Aware**

- Background shading on all time-series charts to show regimes
- Bull = light green, Bear = light red, High-vol = light orange
- Structural breaks marked with vertical dashed lines

## **5. Predictions: Uncertainty Visualization**

- Point predictions shown as solid lines
- Uncertainty bands shown as shaded regions
- Monte Carlo percentiles shown as multiple thin lines

## **6. Ethical Filters: Prominent Badges**

- Each filter gets a "grade" badge
- Pass = Green checkmark, Warning = Yellow exclamation, Fail = Red X
- One-sentence interpretation under each badge

## **G.5) Gemini Workflow Summary**

```
[Complete Company Profile]
  ↓
[Gemini API Call with Structured Prompt]
  ↓
[Gemini Generates Markdown Report]
  ↓
[Post-Processing: Chart Generation]
  ↓
[Post-Processing: Chart Insertion]
  ↓
[Post-Processing: Validation]
  ↓
[PDF Conversion]
  ↓
[Final Bloomberg-Style Investment Report]
```

### **Key Innovation: Ethical-First, Hierarchy-Aware Reporting**

Unlike traditional Bloomberg terminals that bury ethics and emphasize short-term price action, this report:

- **Leads with ethics** - filters are in the executive summary, not an afterthought
- **Respects hierarchy** - in survival scenarios, Tier 1-2 dominate the narrative, not Tier 5 speculation
- **Provides context** - explains why weights shift in different regimes
- **Educes the user** - each filter includes educational content on why it matters universally, not just for Islamic finance

This makes the report valuable for:

- **Islamic investors** - full Sharia compliance analysis
- **Risk-averse investors** - emphasis on survival and cash reality
- **Sophisticated investors** - access to full temporal model outputs and causality networks
- **Beginners** - clear explanations and visual dashboards

## H) Execution environment constraints

You want the code made for **Kaggle**, specifically:

- Use Kaggle secrets for API keys
  - Turn internet toggle on
  - Phase 1 (cache building) runs in one notebook session (~30 min)
  - Phase 2-3 (temporal analysis + burn-out) runs in separate session (~2-3 hours)
  - Phase 4 (prediction + report) runs in final session (~15 min)
- 

## G) Ethical / analysis filters you want in the product

You include a set of filters that change how the user interprets financial signals.

### 1) "Purchasing Power" filter (inflation adjustment)

- Illusion: stock price goes from \$100 to \$105 (+5%).
- Reality: if inflation is 7%, the investor lost 2% purchasing power.
- Universal impact: exposes nominal gains as potentially misleading.
- You want a "Real Return" view that acts like X-ray glasses for the bank account.

### 2) "Solvency" filter (Debt-to-Equity / Riba)

- Illusion: large share price growth.
- Reality: if funded by 3:1 debt-to-equity, the company is fragile.
- A bad month of high interest rates can collapse it.
- Universal impact: beyond religion, it's about stability and quality investing.
- You want high-debt companies flagged as "Fragile" to protect users from likely recession bankruptcies.

### 3) "Gharar vs Probability" filter (volatility index)

- Illusion: a pattern predicts a 20% jump tomorrow.

- Reality: if Beta or volatility is extreme, the jump is as likely to be a 20% drop.
- Universal impact: a “Stability Score” helps separate calculated moves from wild guesses.
- This shifts users away from a gambling mindset regardless of religion.

#### **4) “Cash is King” filter (Free Cash Flow Yield)**

- Illusion: record earnings with no actual cash due to uncollected revenue.
- Reality: “Profit is an opinion, but cash is a fact.”
- Universal impact: Free Cash Flow Yield ensures liquid wealth generation.
- You present this as both Sharia compliance (real business) and elite fundamental analysis.

---

#### **H) Technical “Alpha” protection (anti-reverse engineering / ethics)**

You propose a protection mechanism:

- If you provide the full next-day candlestick series at very high accuracy (example: 98%), a sophisticated user could exploit it for intraday trading, which conflicts with your ethics as a Muslim.
- The mask you propose:
  - Only show a single data point (the **Low**) for next-day charts.
  - Show full charts for next week, month, and year.
  - This hides the intraday “pathway” for traders.

---

#### **I) What you asked to do next (as written in the file)**

You state that the next step is:

- No coding yet.
- Articulate everything.
- Get outlines ready.
- Find the variables and their formulas using Eulerpool structure.

# Developer Guide: Eulerpool “Decision + Linked Variables” 2-Year Cache (Implementation Spec)

## Goal

Build a **daily, point-in-time feature cache** for a selected company over the last **2 years** that includes:

- **Decision variables** (internal company variables).
- **Linked variables** (external / relational / comparative variables).
- Clear labeling of:
  - **Direct**: extracted from Eulerpool endpoints as-is.
  - **Derived**: computed via formulas from extracted fields.
- Cache must be usable for day-by-day temporal modeling **without look-ahead bias**.

This spec is intentionally written so code can be built without relying on knowing every exact JSON key up front.

---

## 0) Definitions

### Decision variables (internal)

- Values describing the company's internal state and internal outcomes.
- Primarily from financial statements + market microstructure.

### Linked variables (external)

- Values describing the environment and network: sector/industry peers, competitors, supply-chain entities, financial institutions, country/macros.
- Some come from Eulerpool (classification + other tickers' data).
- Some come from non-Eulerpool sources (FRED, Gemini relationship discovery, internal graph).

### Point-in-time rule

For a day  $t$ , the feature value must reflect **only what was known on or before  $t$** .

---

## 1) Data acquisition (Eulerpool) — minimum endpoints

Use these endpoint groups (names are conceptual; map them to actual Eulerpool routes used in your account):

## 1. Equity Profile

- Purpose: identity + classification + country/currency.
- Frequency: low (store snapshots; update when changed).
- Example route pattern shown by Eulerpool: `/equity/profile/{isin}`.

## 2. Quotes / OHLCV

- Purpose: daily market series.
- Frequency: daily.

## 3. Income Statement

- Purpose: profitability structure.
- Frequency: quarterly/annual.

## 4. Balance Sheet

- Purpose: solvency + liquidity.
- Frequency: quarterly/annual.

## 5. Cash Flow Statement

- Purpose: cash reality.
- Frequency: quarterly/annual.

## 6. (Optional) Stock Ownership

- Purpose: ownership/float/institutional influence.
- Frequency: periodic.

Implementation note:

- The developer should implement each endpoint with a small “adapter” that outputs a normalized schema (see Section 3).

---

## 2) Cache design (daily table) — required columns

For each trading day `t` in the last 2 years, store a record:

### 2.1 Identity (from Profile)

Store as-of values (repeat daily or join via dimension table):

- `isin`
- `ticker`
- `exchange`
- `currency`
- `country`
- `sector`
- `industry`
- `sub_industry` (if available)

**Type:** Direct

## 2.2 Market data (from Quotes)

Minimum:

- `open`, `high`, `low`, `close`
- `volume`

Recommended if provided:

- `close_adjusted` (or `adj_close`)
- `vwap`
- `market_cap`
- `shares_outstanding` (if daily available)

**Type:** Direct

## 2.3 Statement line items (raw) — store as reported

Store the raw fields you receive from Eulerpool statements.

- Income statement: revenue, gross profit, operating income (EBIT), EBITDA, net income, interest expense (if available), taxes, etc.
- Balance sheet: total assets, total liabilities, total equity, current assets, current liabilities, cash & equivalents, short-term debt, long-term debt, etc.

- Cash flow: operating cash flow, capex, investing CF, financing CF, dividends paid, etc.

**Type:** Direct (but periodic)

## 2.4 Daily “as-of” statement alignment (critical preprocessing)

For each day `t`:

1. Find the latest statement period `p` with `report_date <= t`.
2. Attach all statement values from `p` onto day `t` as:
  - `*_asof` fields.

Example:

- `revenue_asof`
- `total_equity_asof`
- `operating_cash_flow_asof`

**Type:** Preprocessing

---

## 3) Normalization layer (how code “automatically understands what to get”)

Implement a **mapping-driven feature extractor**.

### 3.1 Endpoint adapters

Each adapter returns a normalized dict with canonical names, regardless of Eulerpool’s raw key names.

Example pseudo-interface:

```
profile = fetch_profile(isin)
quotes  = fetch_quotes(isin, start, end)
inc     = fetch_income_statement(isin, start_period, end_per
riod)
bs      = fetch_balance_sheet(isin, start_period, end_perio
d)
cf      = fetch_cashflow_statement(isin, start_period, end_
period)
```

```

profile_norm = normalize_profile(profile)
quotes_norm  = normalize_quotes(quotes)
inc_norm     = normalize_income(inc)
bs_norm      = normalize_balance_sheet(bs)
cf_norm      = normalize_cashflow(cf)

```

### 3.2 Canonical field dictionary

Maintain a single `canonical_fields.yml` (or JSON) that lists:

- Canonical variable name
- Category: decision vs linked
- Source: endpoint name
- Extraction mode: direct vs derived
- Formula (if derived)

Example entries:

- `debt_to_equity`: derived, formula uses `total_debt_asof` and `total_equity_asof`
- `fcf_yield`: derived, formula uses `free_cash_flow_ttm_asof` and `market_cap`

This lets code “know what to get” by reading the dictionary.

## 4) Decision variables (internal) — the required derived set

Compute these daily from direct fields.

### 4.1 Returns & risk (from Quotes)

- `return_1d = close[t] / close[t-1] - 1`
- `log_return_1d = ln(close[t]/close[t-1])`
- `volatility_21d = stddev(log_return_1d, 21)`
- `drawdown_252d = close / rolling_max(close, 252) - 1`

**Type:** Derived

### 4.2 Capital structure (Solvency Filter)

Define:

- `total_debt_asof = short_term_debt_asof + long_term_debt_asof` (or Eulerpool total debt if provided)

Compute:

- `debt_to_equity = total_debt_asof / total_equity_asof`
- `net_debt = total_debt_asof - cash_and_equivalents_asof`
- `net_debt_to_ebitda = net_debt / ebitda_ttm_asof`
- `interest_coverage = ebit_ttm_asof / interest_expense_ttm_asof` (if interest expense exists)

Type: Derived

### 4.3 Liquidity / survival mode signals

- `current_ratio = current_assets_asof / current_liabilities_asof`
- `quick_ratio = (cash_and_equivalents_asof + receivables_asof + short_term_investments_asof) / current_liabilities_asof` (if components exist)
- `cash_ratio = cash_and_equivalents_asof / current_liabilities_asof`

Type: Derived

### 4.4 Cash reality (Cash is King Filter)

Define:

- `free_cash_flow = operating_cash_flow_asof - capex_asof`

Compute:

- `free_cash_flow_ttm_asof` (sum last 4 quarters, point-in-time)
- `fcf_margin = free_cash_flow_ttm_asof / revenue_ttm_asof`
- `fcf_yield = free_cash_flow_ttm_asof / market_cap[t]`

Type: Derived

### 4.5 Profitability quality

- `gross_margin = gross_profit_ttm_asof / revenue_ttm_asof`

- `operating_margin = ebit_ttm_asof / revenue_ttm_asof`
- `net_margin = net_income_ttm_asof / revenue_ttm_asof`
- `roe = net_income_ttm_asof / avg_equity_asof`

**Type:** Derived

## 4.6 Valuation (optional but recommended)

- `pe_ratio_calc = market_cap[t] / net_income_ttm_asof`
- `earnings_yield_calc = net_income_ttm_asof / market_cap[t]`
- `ps_ratio_calc = market_cap[t] / revenue_ttm_asof`
- `enterprise_value = market_cap[t] + total_debt_asof - cash_and_equivalents_asof`
- `ev_to_ebitda = enterprise_value / ebitda_ttm_asof`

**Type:** Derived

Implementation note:

- If Eulerpool provides ratios directly (P/E, ROE, fcf\_yield, debt\_to\_equity), store them too as `*_direct` and keep `*_calc` as validation.

## 5) Linked variables (external) — what to compute and what to store

Linked variables are computed from:

- Peer group aggregates (sector/industry)
- Explicit relationship sets (competitors, supply chain, institutions)
- Country/macros (FRED)

### 5.1 Sector/industry peer aggregates (computed by you)

For each day `t` and each peer set `s` (sector or industry):

- `sector_median_return_21d[t]`
- `sector_median_vol_21d[t]`
- `industry_median_debt_to_equity[t]`
- `industry_median_fcf_yield[t]`

Then company-relative linked variables:

- `rel_strength_vs_sector = return_21d_company - sector_median_return_21d`
- `valuation_premium_vs_industry = pe_ratio_calc - industry_median_pe`

**Type:** Derived (requires downloading data for peer tickers too)

## 5.2 Competitor / supply chain / financial institution aggregates

Inputs:

- Relationship lists are discovered by Gemini and stored as a graph:
  - `competitors[]`
  - `suppliers[]`
  - `customers[]`
  - `financial_institutions[]`

For each relationship list `R`, compute daily aggregates:

- `competitors_avg_return_21d`
- `supply_chain_avg_drawdown_252d`
- `fin_inst_avg_vol_21d`

**Type:** Derived (Eulerpool provides data for each related entity once you have the list)

---

## 6) Cache rules (no look-ahead + missing data)

### 6.1 No look-ahead enforcement

- Statements must be aligned by **report date  $\leq t$** .
- Do not use “latest” values for historical dates unless the API is explicitly point-in-time.

### 6.2 Missingness handling

For any variable that cannot be computed on day `t`:

- Store value = `null`

- Store a flag = `is_missing_<var> = 1`

Do not forward-fill market data across missing trading days.

Forward-fill statements is allowed only through the `as-of` alignment logic.

---

## 7) Deliverables from the extraction module

The developer should implement:

1. `build_company_cache(isin, start_date, end_date) -> daily_dataframe`
2. `build_linked_cache(isin, relationships, peer_sets, start_date, end_date)`
3. `compute_derived_features(daily_dataframe) -> daily_dataframe`
4. `validate_features(daily_dataframe)`
  - checks: division-by-zero, unit consistency, negative equity edge cases, etc.

## 8) Notes about survival mode inputs

Survival mode uses a combination of decision + linked variables:

- Decision: liquidity ratios, cash runway proxies, leverage, cash flow health.
- Linked: country protection/survival from FRED, plus peer stress.

The survival-mode decision must be computed per day (or per week) and stored as:

- `company_survival_mode_flag`
  - `country_survival_mode_flag`
  - `country_protected_flag`
- 

## 9) What is NOT covered by Eulerpool (must come from elsewhere)

- Inflation series for real return (usually macro source such as FRED).
  - Country protection logic inputs (you explicitly plan FRED).
  - Competitor/supply-chain discovery (Gemini + internal graph).
- 

## 10) Minimal variable list to ensure the four filters work

To satisfy your explicit filters, ensure these are available daily:

- Purchasing Power filter: `nominal_return_1d`, `inflation_rate`, `real_return_1d`
  - Solvency filter: `debt_to_equity` (calc), plus debt/equity components
  - Gharar/Probability filter: `volatility_21d` and optionally `beta_252d`
  - Cash is King filter: `free_cash_flow_ttm_asof`, `market_cap`, `fcf_yield`
- 

## 11) FRED variables to support Operator 1 (macro linked variables)

Below is the **minimum macro variable set** the code should be able to pull from FRED (per country when possible). The implementation must be **config-driven**, because FRED series IDs differ by country.

### 11.1 Purchasing power / inflation (required)

Used to compute real returns and avoid nominal-gain illusion.

- `inflation_rate` (monthly or YoY)
- `inflation_rate_daily_equivalent` (derived conversion)
- `real_return_1d = nominal_return_1d - inflation_rate_daily_equivalent`

### 11.2 Rates & monetary conditions (required for survival-mode + fragility)

Used to model debt stress and environment tightening.

- `policy_rate` (central bank policy rate proxy)
- `short_rate` (e.g., 3M rate)
- `long_rate` (e.g., 10Y yield)
- `yield_curve_slope = long_rate - short_rate`

### 11.3 Currency & external pressure (recommended)

Used to detect imported inflation / FX stress for non-USD countries.

- `usd_fx_rate` (local currency per USD) or a close proxy
- `fx_volatility` (derived from FX series)

## 11.4 Credit stress / financial system stress (recommended)

Used to detect country survival mode and banking stress.

- `credit_spread` (corporate spread proxy when available)
- `financial_conditions_index` (if available)
- `bank_stress_proxy` (config-driven; may be missing for many countries)

## 11.5 Real economy health (recommended)

Used to detect demand collapse / recession / survival environment.

- `unemployment_rate`
- `gdp_growth` (quarterly)
- `industrial_production` (monthly)
- `cpi_level` (if inflation rate is not directly available)

## 11.6 Core config files (developer requirement)

Add a FRED mapping layer:

- `config/fred_series_map.yml` maps `country_code` → series IDs for the above canonical variables.
- `config/country_protection_rules.yml` defines how to compute `country_protected_flag`.
- `config/country_survival_rules.yml` defines how to compute `country_survival_mode_flag`.

## 11.7 Missing series policy (must follow)

If a series is missing for a country:

- Store the canonical variable as `null`.
- Store `is_missing_<var> = 1`.
- Continue pipeline (no hard failure).