ChatGPT

Analysis of the ATFT-GAT-FAN Stock Prediction System

Current Model Architecture and Suitability for Stock Forecasting

The core model **ATFT-GAT-FAN** (Adaptive Temporal Fusion Transformer + Graph Attention Network + Frequency Adaptive Normalization) is a cutting-edge architecture tailored for financial time-series forecasting [1]. It blends a **Temporal Fusion Transformer (TFT)** backbone with a **Graph Attention Network (GAT)** and specialized normalization layers: - **Temporal Fusion Transformer (TFT)** – processes each stock's historical features over a sequence (e.g. 60 days) with LSTM and multi-head attention components, enabling the model to capture temporal patterns and feature importances over time [2] [3]. This is well-suited for stock data, as it can learn time-dependent signals (trends, mean-reversion, seasonality) and gives interpretability via attention on features and time steps. - **Graph Attention Network (GAT)** – models relationships between different stocks. By treating stocks as nodes in a graph (with edges defined by correlations or sector links), the GAT allows information to flow between related stocks [4] [5]. This is highly relevant in equity markets where stocks are not independent (e.g. industry peers or index components move together). The architecture applies GAT on the **last time-step's hidden representations** of each stock, then concatenates graph-derived embeddings with the TFT output [5] [6]. This helps the model adjust a stock's prediction based on its peers' behavior, potentially improving performance during sector-wide moves or market events. - **Frequency Adaptive Normalization (FAN)** and **Slice Adaptive Normalization (SAN)** – custom normalization layers that adapt to different frequency components of the data [7] [8]. These likely help the model handle multi-scale temporal patterns (e.g. daily vs. weekly cycles) by normalizing features across various rolling window lengths. This is an innovative choice for financial data which often has seasonalities or cyclical behavior at different frequencies. - **Variable Selection Network (VSN)** – a feature gating mechanism that learns to weight each input feature's contribution at each time step [9]. This aligns with the TFT design and is valuable given the large number of input features; it can ignore irrelevant features dynamically to reduce noise. In financial terms, this acts like an attention mechanism on features, potentially improving generalization by sparsifying unimportant inputs (the implementation also includes a sparsity loss to encourage this [10]).

Overall, the architecture is **highly suited to stock forecasting** because it addresses key complexities: temporal dynamics, cross-sectional relations, and feature selection. By combining these components, ATFT-GAT-FAN achieved about **20% improvement in 1-day ahead rank correlation (RankIC)** over a prior baseline [11]. This indicates that the architecture is effectively capturing additional signal. The multi-horizon output design (predicting returns at 1, 5, 10, 20 days into the future) is also appropriate, as it allows the model to learn both short-term and medium-term predictive patterns in one framework [12]. The model's output head produces quantile forecasts for each horizon (enabling uncertainty estimates), and uses a custom multi-horizon **Huber quantile loss** for robust training [13]. All these are best practices in forecasting volatile financial targets.

Bottlenecks/Challenges: The sophistication of ATFT-GAT-FAN comes with increased complexity. One consideration is that the **Graph module is currently limited to 50 nodes** (as per config) for tractability [14], meaning only a subset of stocks (perhaps the largest 50) form the graph. While this improves efficiency, it might ignore relationships outside this subset. Important cross-stock information could be

missed for the remaining universe, or the graph might not fully reflect all market relationships. Scaling GAT to hundreds of stocks is non-trivial (computationally expensive), but exploring techniques like **sparser graphs or clustering** (e.g. group stocks by sector and use sector-aggregated nodes, or use nearest-neighbor graphs instead of fully connected correlations) could help include more stocks' information. Another potential bottleneck is the **sequence length (60 days)** [15] – this covers ~3 months of history, which may miss longer-term trends or regime shifts. The current TFT uses LSTM internally [3]; transitioning to a full Transformer-based encoder (with positional embeddings) could enable longer sequence modeling if needed, though at a cost of more compute. Additionally, the complex architecture might be prone to overfitting if not regularized enough, given the modest amount of data (on the order of 600k samples covering ~600 stocks over a few years). The team has mitigated this with various regularization techniques (dropouts, gating, normalization), but complexity remains something to monitor. In a production setting, complexity also affects **model interpretability and debugging** – while TFT and GAT individually are somewhat interpretable (via attention weights and gates), the combined model's decisions might be harder to explain to stakeholders. Despite these challenges, the design is **state-of-the-art** for time-series forecasting, and largely appropriate for the stock prediction task. It brings a rich representational capacity that, if utilized well, can capture subtle patterns that simpler models (like traditional tree ensembles or basic LSTMs) might miss.

Feature Engineering Pipeline and Data Sources

The system's feature pipeline is extensive and integrates **diverse data sources** from the Japanese market via the J-Quants API. The raw inputs include daily price data and a trove of engineered features, organized into categories:

- **Basic Price/Volume Features (6 columns):** Daily OHLCV (Open, High, Low, Close, Volume) and an index column [16].
- **Technical Indicators (~60 columns):** A selection of technical analysis features derived from price and volume. According to documentation, these were chosen from a pool of 713 indicators, focusing on trend, momentum, volatility, and volume signals [17] [18]. Examples include moving averages (SMA 5/10/20/50), exponential moving averages (EMA 12/26), MACD & signal, ADX (trend strength), RSI (momentum), Stochastic %K/D, Williams %R, ROC, Bollinger Bands (volatility), ATR, VWAP, OBV, CMF, etc [19] [20]. These features help the model detect classic trading signals (e.g. overbought/oversold from RSI, trend direction from moving averages) in an automated way.
- **"Quality" Features (+6 columns):** These are meta-features added to improve data quality and stationarity [21]. For instance, **cross-sectional quantile ranks** (ranking a stock's feature relative to all other stocks on the same day) and **sigma-threshold outlier flags** are computed [21] [22]. Such features can highlight how unusual a stock's move or valuation is in the cross-section (e.g. a high return compared to peers might indicate news). They serve to normalize and filter extreme values, helping the model not be skewed by outliers or by differences in scale between stocks.
- **Event & Sentiment Features:** The pipeline taps into advanced datasets like **earnings announcements, short-selling positions, and listed info** (Phase 1 features), and even more advanced Phase 2 features:
- **Margin Trading Features** – signals from margin trading data (investors borrowing to buy/sell stocks). For example, **margin balance ratios**, changes in margin buy/sell positions ("velocity"), divergence between long vs short margin activity, and an overall stress indicator combining these [23]. These come from daily and weekly margin interest reports via J-Quants API [24]. They provide insight into leveraged trading behavior, which can foreshadow price pressure (e.g. if many are margin-long a stock, forced liquidations could cause declines).
- **Option Market Sentiment** – derived from Nikkei 225 index options data, e.g. **put/call ratio** (a fear gauge), implied volatility skew (difference between put and call IVs indicating tail-risk sentiment),

open interest imbalances, average implied vol, term structure slope, etc [25]. These features reflect broader market sentiment and risk appetite, which can trickle down to individual stocks. They are fetched via an index options API [26].

- **Institutional Flow Features** – metrics from the `TradesSpec` API capturing net buying/selling by investor categories [27]. Features like **institutional_accumulation**, **foreign_vs_domestic divergence**, and **flow concentration indices** measure how different investor segments (institutions, foreigners, retail, etc.) are behaving [27]. Unusual accumulation by "smart money" or persistent buying by institutions could be predictive of future performance.
- **Short Selling and Other Market Metadata:** Although not explicitly detailed in the snippet above, Phase 1 included short-selling position features and listed company info [28] [29]. These likely cover daily short balances and perhaps fundamental info like market cap or sector classification. Earnings calendar events might also be part of Phase 1 (e.g. a feature indicating days around earnings announcements).

All these features are integrated by the **MLDatasetBuilder** into a single panel data table, with careful alignment. Notably, weekly data (like weekly margin) are forward-filled and merged in an "as-of" manner so that at any given date, only information available up to that date is used [30] [31]. For example, a weekly margin report published on Monday (for the previous week) is assigned to subsequent days with a lag, ensuring no forward leakage [32]. Similarly, daily margin data is merged with a T+1 lag (published today, usable tomorrow) [33]. The pipeline explicitly handles these **as-of joins** to avoid lookahead bias.

After feature assembly, a **Cross-Sectional Normalization** is applied: each day's features are normalized (e.g. z-scored) across the cross-section of stocks [34]. Importantly, this normalization is fit using only the training data to prevent information leaking from future test data [35]. This step yields features that have mean ~0 and unit variance each day, which is crucial in a stock context to remove market-wide moves (e.g. if all stocks are up 5% on a day, the model should focus on relative differences, not the market drift). The pipeline uses a **Polars** (Rust-based DataFrame library) implementation for this, which is 3-5x faster than pandas [36]. Outliers beyond 5 sigma can be clipped as part of this robust scaling [37].

Finally, the data is split for modeling. The project employs a **walk-forward time-split** strategy with an embargo: e.g. 5 chronological folds, each with a 20-day gap between train and test periods to ensure no overlap of information [38] [39]. This means the model is always validated on out-of-sample future periods, simulating a live trading scenario. It prevents any form of leakage that could occur if, say, training and test sets had adjacent dates (the embargo gap avoids training on data that is too close to the test period) [40].

Data Sources & Pipeline Evaluation: The data pipeline is designed for **production robustness**. It uses an asynchronous fetcher (capable of ~150 parallel API calls) to ingest data quickly [41] [42], and stores raw data in an object storage (MinIO) and processed data as Parquet files for efficient reloads [43]. There are layers of validation (ensuring coverage >98%, checking for duplicates or high missing rates) [44] [45]. Performance-wise, the team reports the entire pipeline (over ~606k samples × 139 features) executes in ~1.9 seconds in a production setting [46] – a testament to its optimizations (using Polars lazy operations, column projection to only load needed columns, and memory mapping) [47]. This means daily feature updates and model retraining can be done very quickly, an important requirement for real-world deployment.

Possible Gaps and Opportunities in Features: The feature set is already rich and domain-relevant. One possible area to explore is **fundamental indicators** (if not already included). For example, valuation metrics (P/E, P/B ratios), growth metrics (earnings growth, analyst forecasts), or balance sheet indicators could add a longer-term signal component to complement the predominantly technical and flow features. Since the J-Quants API includes financial statements and analyst forecasts, features like

“earnings surprise” or “revisions in forecasts” might enhance the model’s ability to predict medium-term moves around fundamentals. Another avenue is **alternative data** – e.g. news sentiment or web search trends for companies, which could proxy retail interest or public sentiment. These data weren’t mentioned in the current pipeline (likely due to focus on J-Quants sources), but in production such data could improve robustness (especially for event-driven stock jumps).

Additionally, while the model has many features, not all may be equally useful. The config indicates a feature selection step (e.g. mutual information-based selection of top 100 features) is possible [48], though the model’s internal VSN also performs feature selection on the fly. It may be beneficial to periodically **re-evaluate feature importance** (through retraining or using the model’s gating signals) and prune or down-weight stale features. This keeps the feature set lean and focused, reducing noise and overfitting risk. Given that the pipeline initially had 139+ features which grew to ~160 after adding margin features [49], ensuring that this high-dimensional input truly adds signal is important. The team’s use of gating and regularization helps, but an offline analysis (e.g. ablation tests) to drop certain feature groups could uncover if any feature category is actually detrimental or redundant. In fact, an `ablation_report.py` script exists, suggesting they are already comparing variants like adding interactions, winsorizing outliers, etc., to quantify feature contributions.

In summary, the feature engineering is **comprehensive and thoughtfully designed**. It leverages multiple data streams (price, volume, sentiment, institutional flows, etc.) and addresses data alignment, normalization, and splitting in a manner consistent with real-world constraints. This provides a strong foundation for the model. Future improvements could involve incorporating new data sources (fundamental or alternative data) and continuously refining the feature set for maximum signal-to-noise ratio.

Training and Evaluation Methodologies

The training approach for ATFT-GAT-FAN follows rigorous methodologies to ensure the model’s performance claims are reliable and the training process is stable:

- **Walk-Forward Cross-Validation:** As mentioned, model evaluation uses purged walk-forward splits with an embargo [38]. Practically, this means the team trains the model on an initial period, tests on the next period, and then rolls forward, repeating this for 5 folds (by default). Performance metrics are averaged across folds. This strategy mirrors how the model would be used in production (always predicting forward in time) and provides a realistic gauge of generalization. It also helps in hyperparameter tuning – rather than relying on a single train/val split, they can ensure the model isn’t overtuned to one particular period. The embargo (20 days gap) further solidifies the evaluation by avoiding any subtle temporal overlap [39].

- **Metrics Tracked:** The primary metrics of interest are domain-specific: **Rank IC (Rank Information Coefficient)**, which is the Spearman rank correlation between the model’s predicted scores and actual next-day returns, and related statistics like IC’s t-stat (ICIR) and **hit rate** (probability of correct directional prediction) [50]. They also compute **Sharpe ratio** of a simple long-short strategy based on the model (e.g. long top decile, short bottom decile each day) [50]. These metrics directly measure the financial value of the model’s predictions – e.g., a higher RankIC means the model’s rankings of stocks are more aligned with realized outcomes, which is crucial for a quantitative strategy. The evaluation protocol even mentions using bootstrap confidence intervals and significance tests (Diebold–Mariano) to verify improvements [51]. This level of statistical rigor ensures that any claimed improvement (like the +20% RankIC) is not just a fluke.

- **Training Loss and Objectives:** The model is trained to predict multiple quantiles of future returns. The loss function is a **Quantile loss** (Pinball loss) for these predictions, which was further made robust by using a Huber formulation for the main horizon [13]. In effect, the loss is less sensitive to large outlier errors (important in finance where occasional crashes or spikes can otherwise dominate training). Moreover, the config shows "robust_mse" with specific horizon weights [52] – they put higher weight on the 1-day horizon vs. longer horizons (for example 1-day has weight 1.0, 5-day 0.7, 10-day 0.5, etc.). This is a strategic choice: short-term predictions are given more emphasis during training, aligning with the goal of strong near-term RankIC (since those correlate with daily trading gains). Yet, by including longer horizons (5d, 10d, 20d) in a multi-task setup, the model can share some learned patterns across horizons and not overfit solely to one-day noise. This multi-horizon training is beneficial as it forces the model to also account for intermediate trends (5-day, 1-month returns) to some extent, potentially improving its robustness.

- **Optimization and Regularization:** Training uses the AdamW optimizer with weight decay for generalization [53]. A cosine learning rate schedule with warmup is employed for a smooth training curve [54]. Notably, the batch size is large (up to 2048), and **"day batching"** is enabled [55] – meaning the data loader likely batches by date, feeding the model all stocks of a given day as one batch. This is clever because it aligns with how the graph is constructed (the GAT can operate on all stocks in the batch with known edges for that day). It also stabilizes training by reducing random mixing of dates; essentially each gradient step looks at one day's cross-sectional data. To compensate for any single-batch bias, they shuffle the order of days during training. The training loop is managed via PyTorch Lightning (the model subclassing `LightningModule`), which handles things like mixed precision (they use bf16 mixed precision for speed) [56] [57] and gradient clipping (set to 1.0 to prevent exploding gradients) [56].

- **Advanced Training Techniques:** The team implemented several advanced tricks to improve training stability and performance:

- **EMA Teacher Model:** An Exponential Moving Average of the model weights is maintained as a "teacher" network [58]. This teacher can be used to generate targets or to periodically align the student model. In practice, EMA smoothing of weights often yields a more stable version of the model for evaluation (less noisy than the latest weights). The config shows `use_ema_teacher: true` with decay 0.999 [59]. This suggests that during training, they may either (a) use the EMA model's predictions as an additional consistency target, or (b) simply use the EMA model for validation and final predictions (which usually improves generalization). This is a proven technique in semi-supervised learning and model stabilization.
- **LayerScale Initialization:** They apply **LayerScale** to certain layers (multi-head attention and MLP layers) – essentially initializing those layer weights to a smaller scale (like 0.1) [60]. By shrinking initial weights in the output layers, the model starts in a near-linear regime, which helps prevent unstable learning dynamics in deep transformers. The config confirms `use_layerscale: true` with init 0.1 [59], and the training script explicitly multiplies some weights by this factor [61]. This likely contributed to the improved training stability and the increase in prediction variance (Pred.std) observed [62] (the model is no longer collapsing to very small output changes).
- **Gradient Checkpointing:** The improvements list mentions gradient checkpointing (saving memory by recomputing activations on the fly) [58]. While the config currently has it off by default [63], it can be enabled if memory becomes a bottleneck, allowing larger models or longer sequences without OOM errors. They also built an **OOM auto-recovery** mechanism [64] – likely a wrapper that catches out-of-memory errors and retries with smaller batch or accumulated gradients, ensuring long training jobs don't crash unexpectedly.

- **Parameter Group Scheduling:** Different parts of the network can have different learning rates or weight decay. For instance, one might use a lower LR for the transformer backbone and a higher LR for a newly added output layer. The team indicates they added **layer-wise learning rate scheduling** [58]. In practice, this kind of fine-tuning can help if certain submodules learn at different paces. (The exact config for this isn't visible, but it's noted as implemented).
- **Curriculum Learning:** A curriculum scheduler is present to adjust training focus over epochs [65] [66]. For example, in early epochs the model might focus on simpler patterns or a subset of horizons, and later incorporate all targets. The code snippet shows phases with horizon weighting adjustments [67]. Curriculum learning can be useful in financial data to let the model first learn broad signals before fitting subtle details. It's optional, but can be enabled to potentially improve convergence.

- **Early Stopping and Baseline Comparison:** They monitor validation metrics and use early stopping with patience (15 epochs) on improving RankIC@5d [68]. Interestingly, they integrate a baseline LightGBM model training in their pipeline and require the neural model to beat this baseline by at least +10% before considering it successful [69] [70]. This "baseline feedback" is a great practice – it ensures the complex model is justified. If the simpler model performs similarly, one might question deploying the complex one. In this case, the ATFT-GAT-FAN clearly surpasses the baseline (e.g. RankIC 0.18 vs baseline 0.15 is +20% [11], exceeding the +10% gate). Early stopping can also incorporate this baseline (stop if no improvement relative to baseline), which they have toggled in config [71].

- **Monitoring and Logging:** For a production-level system, observability is key. The training integrates **TensorBoard and Weights & Biases (W&B)** logging [64]. Metrics like training/validation loss, IC, Sharpe are tracked per epoch [72]. They have also set up automated alerts – for example, if a metric dips or if the training stops unexpectedly, the system can notify the team [64]. This kind of monitoring is critical when models are retrained periodically in production; it helps catch issues early (like data feed problems or model divergence).

In summary, the training methodology is **robust and aligned with best practices for time-series ML in finance**. By using walk-forward CV, appropriate loss functions, and a suite of regularization/stability techniques, they maximize the chances that the model's performance will hold up in real deployment. The fact that they improved training speed (~6.7% faster) and reduced memory usage (~12% less memory) in the latest iteration shows an eye for efficiency as well [73]. One area to ensure going forward is that the training data remains up-to-date – for production, one would retrain or fine-tune the model as new data comes in (e.g. monthly or weekly retraining). Given the pipeline speed (sub-2 seconds) and model training time on GPU (possibly on the order of a few hours for 75 epochs), this cadence is feasible. They also might consider **online updating** or warm-starting from previous weights when doing frequent retrains, to shorten training time while adapting to new regimes.

Current Performance Metrics and Gaps

Performance Achieved: Based on the documentation and team's reports, the improved ATFT-GAT-FAN model delivers significantly better predictive performance than previous versions. The key metric **RankIC @ 1-day** improved from ~0.150 to ~0.180 (+20%) [11]. In practical terms, a RankIC of 0.18 means the model has a decent ability to rank stocks by next-day returns – for context, in quant finance an IC of 0.1–0.2 is considered a strong signal (because stock returns are noisy). They also report similar ~20% relative improvements for 3-day and 5-day horizons (e.g. 5-day RankIC from 0.095 to 0.115) [12]. This indicates the model is adding value even a week out, though performance naturally declines with horizon. The

average loss (perhaps quantile loss or MSE) dropped by 10% [74], confirming better accuracy in a traditional sense as well.

The **Sharpe ratio** of the model's hypothetical strategy isn't explicitly in the snippet, but the config suggests they were targeting ~0.8–0.85 Sharpe [75]. If RankIC is ~0.18, an information ratio (ICIR) can be computed by scaling by sqrt(N) over some period; the values likely imply an ICIR that's solid. They also mention **hit rate and portfolio Sharpe** are evaluated [50]. If the model's predictions were used to form a daily long-short portfolio, the Sharpe likely lies in the 1.0 range (just speculation based on IC, as a 0.18 IC could correspond to ~55-60% hit rate and a reasonable long-short return spread). One specific metric they tracked is the **"Pred.std ratio"** – this went from 0.80 to 0.85 [62]. This ratio compares the standard deviation of the model's predicted returns to the actual returns' std. A value of 0.85 means the model's predictions have somewhat lower volatility than actual outcomes. The increase toward 0.85 (from 0.80) is actually a positive sign – it suggests the model's predictions became more bold or varied after improvements (previously they might have been too conservative/clustered around the mean). Ideally, you want predictions that have similar dispersion to real returns, otherwise the model might be underconfident. The improvements like LayerScale likely helped boost this by preventing the model from collapsing its outputs.

Identified Gaps: Despite the strong performance, there are areas to probe for further improvement: - The **absolute level of RankIC (0.18)**, while good, might still be below what's desired for production alpha. The config notes a goal of 0.20 RankIC [75]. Pushing from ~0.18 to >0.20 could yield a disproportionate gain in portfolio performance (due to compounding effects and being able to take larger positions with confidence). This gap might require more data or model enhancements since 0.18 likely already reflects a lot of signal extraction from available features. - **Stability over time:** It's important to ensure the model's performance is consistent in different market regimes. The walk-forward results give an average; we should check if, for example, the model did equally well in calm periods vs. volatile periods. If not already done, one could analyze performance in sub-periods (the ablation script hint had options for time-slice validation). If any regime is a weak spot (say, during market crashes the IC dropped or turned negative), that's a gap to address – possibly via regime-specific modeling (discussed below in recommendations). - **Longer horizon performance:** The 10-day and 20-day horizon metrics weren't explicitly given in the snippet, but likely the RankIC at 20d is quite low (maybe near 0, as far future returns are very noisy). If the business goal includes longer-term forecasts (for say a monthly rotation strategy), the current setup might need augmentation. The multi-horizon training tries to balance this, but perhaps separate models or features are needed for longer horizons (e.g. fundamental data for 1-3 month predictions). - **Complexity vs. incremental benefit:** The team added many innovations (GAT, advanced features, normalization tricks). Each adds some accuracy but also potential points of failure or overfitting. For production, it's crucial that each component is justified. For instance, if we ablate GAT and still get ~0.175 RankIC, one might decide the complexity of maintaining graph data in production is not worth a tiny +0.005 IC. We don't have the ablation results here, but that analysis is important. A bottleneck could be that GAT requires computing a correlation graph for the universe daily (the pipeline mentions a 50-node, 266-edge graph built presumably via correlation analysis [76]). If the graph adds significant operational overhead and only marginal gain, that's a gap where simplification or alternative methods could be sought. However, assuming the +20% improvement cited is cumulative of all enhancements, GAT and others likely do contribute meaningfully. - **Real-time inference and data latency:** The pipeline and model seem geared to end-of-day batch predictions (which is typical for daily trading strategies). If there's a need to predict intraday or on streaming data, there may be a gap in current design. Features like margin and flows are mostly daily. The model size (hidden dim 256, 6 layers, GAT, etc.) is not lightweight, but inference on ~600 stocks * quantiles should still be under a second on GPU, which is fine for daily use. If latency were a concern, one might need to consider model distillation or simpler surrogate models for faster inference. Currently this doesn't appear to be an issue unless the production requirement changes.

In summary, **the model's current performance is strong** and has met all improvement targets set by the team (RankIC +20%, loss -10%, etc. all achieved [77]). The remaining gaps are about pushing the frontier further and ensuring robustness: reaching that 0.20 RankIC goal, maintaining performance in all conditions, and balancing complexity vs. benefit for each component. These observations inform the following recommendations.

Recommendations for Production-Level Improvement

To elevate the system to top-tier production performance (both in accuracy and robustness), we propose several enhancements and optimizations, structured by area:

1. Model Architecture Enhancements

- **Leverage Ensembles of Models:** Ensemble methods could substantially boost stability and accuracy. Since the system already allows ensembling checkpoints [78], one approach is to train multiple instances of the model (with different random seeds or slightly varied hyperparameters) and average their predictions. Ensembling tends to reduce variance and improve the consistency of predictions (higher ICIR). For example, ensembling the top 3 models from different folds or runs can yield a higher RankIC than any single model. The config has an `ensemble` option (currently disabled) – enabling this in production (even a simple average of the last few epoch checkpoints or a snapshot ensemble) could push RankIC closer to that 0.20 mark without needing new data [78]. It will increase inference cost, but since it's end-of-day, a small ensemble (3-5 models) is still feasible.
- **Introduce Model Specialization (Mixture of Experts):** The architecture already contemplates a **Regime Mixture-of-Experts** head (RegimeMoE) [79] [80]. We recommend fully utilizing this to handle distinct market regimes. For instance, one expert sub-model could specialize in trending markets, another in mean-reverting or high-volatility markets. The regime features (like market volatility index, trend indicators) can feed a gating network that chooses which expert to trust [81]. Currently the model either uses a multi-horizon head or a RegimeMoE if configured. Enabling and fine-tuning the RegimeMoE with well-chosen regime labels (e.g. bull vs bear market, or high vs low volatility regimes) can improve performance during extreme conditions. This will make the model more robust across cycles – a key for production since markets change character over time. If implementing MoE is complex, an alternative is to **train separate models for different regimes** offline (say, one model on data from calm periods, one on crises) and ensemble them at inference based on current regime detection.
- **Dynamic Graph and Relation Modeling:** If the GAT is adding value, consider expanding its scope. Right now it uses a static correlation-based graph (with threshold 0.3) on 50 nodes [14]. One improvement is to make the graph **dynamic** – e.g., update correlations rolling or use a recent window so the graph reflects current market relationships (they might be doing this already in the daily build). Also, including more nodes or connections could help: for example, use sector or industry groupings as additional edges (all stocks in same sector connected). This injects known relationships that pure correlation might miss. Another idea is a **two-tier graph**: first connect all stocks to their sector ETF or index (capturing broad market move), and second connect stocks with high residual correlation. This way, important global factors are in the graph. If the 50-node limit is due to performance, one can implement **neighbor sampling for GAT** (as done in GraphSAGE models) – the model could sample a subset of neighbors for each node per batch, rather than require a fixed small graph. This would allow scaling to the full universe of stocks gradually. Care must be taken to ensure consistency, but frameworks like PyG support such sampling. The aim is to capture cross-sectional effects beyond the top 50 stocks, as mid- and small-cap stocks might also have predictive links (for example, if a small supplier always moves

after its large customer stock moves – a relationship not captured if that small stock isn't in the top 50).

- **Experiment with Alternative Sequence Models:** While TFT (which uses LSTM internally) is performing well, the field of time-series is moving towards transformer-based models (e.g. Informer, Autoformer) for capturing long-range dependencies. A potential improvement is a **pure attention-based encoder** that can handle longer lookback periods (say 120 days or more) without the memory bottlenecks of LSTM. The current sequence length (60) might miss patterns like yearly seasonality or multi-quarter cycles. A transformer with sparse attention or other efficient architectures could allow extending to longer sequences, potentially improving the model's understanding of long-term trends or mean reversion. We suggest running experiments to compare the TFT (LSTM+attention) vs. a full Transformer encoder on the same data. If the transformer yields even a small boost in IC or better multi-horizon consistency, it might be worth incorporating. That said, pure transformers may require more data to avoid overfitting, so this should be done carefully (perhaps pre-training on unsupervised tasks or using transfer learning from related data could help).
- **Model Simplification for Maintainability:** On the flip side, it's worth evaluating if any components can be simplified **without sacrificing performance**. For instance, if the **Frequency Adaptive Normalization** or **Slice Normalization** layers' effect is marginal, one might remove or simplify them to reduce complexity. Production-level reliability often benefits from fewer moving parts. The improvements doc noted increased code complexity as a downside [82]. Each extra module (FAN, SAN, etc.) is another piece to test and maintain. We recommend conducting ablation tests (disabling one component at a time) to see the impact on validation IC. If a component yields <0.5% IC gain, it might not justify the added complexity in a live system. In those cases, consider dropping it or replacing it with a standard alternative (e.g. if FAN is not crucial, use a simple LayerNorm). This will make the model more **streamlined and interpretable** in production, focusing only on truly value-adding parts. The overall goal is to keep the architecture as **simple as possible, but no simpler** for the task at hand.

2. Feature and Data Improvements

- **Incorporate Fundamental and Macro Features:** As mentioned, adding fundamental data could improve longer-horizon forecasts. Features like earnings yield, revenue growth, profit margins, debt ratios, etc., for each company might help the model identify fundamentally undervalued or overvalued stocks that might appreciate or mean-revert over a month or quarter. Likewise, incorporating **macro indicators** (interest rates, FX rates, economic indices) can provide context – for example, including the USD/JPY exchange rate trend could be vital for export-driven Japanese stocks' performance. If the model is aware of these macro trends, it might better adjust predictions (like, if yen is weakening consistently, export stocks might have a tailwind). Many quant models include macro vars as features for this reason. These could be fetched from external sources (economic data APIs) and merged by date.
- **Alternative Data (Sentiment/NLP):** To capture market sentiment more directly, one can integrate **news and social media sentiment**. For instance, a daily sentiment score for each stock (derived from news headlines or stock forum discussions) could flag bullish or bearish sentiment that isn't yet reflected in technicals. Modern NLP can quantify sentiment or detect when a company is frequently in the news (possibly indicating a catalyst). This kind of unstructured data feature could complement the structured numeric features. In production, it requires a pipeline to fetch and process text data daily, which is non-trivial, but the payoff could be more timely signals (especially around events). Since the system already handles multiple data sources, extending it to an NLP-based feature generator (even a simple one counting news articles or using a pre-trained sentiment model) is feasible.

- **Feature Interaction and Nonlinear Features:** The current feature set is mostly linear transformations of base data (indicators, ratios, etc.). Non-linear interaction features might be useful – e.g., the product of a valuation metric and a technical indicator could capture "value + momentum" interaction effects. The pipeline does mention interaction features (the prefix `x_` features) [83] . Ensuring a rich set of **cross-features** (like sector-relative returns times volume surge, or margin trading change times price momentum) can allow the model to detect conditional patterns. The caveat is that throwing in too many interactions can blow up feature count and noise. A guided approach is to use domain knowledge to create a handful of interaction terms that are known in quant research (for example, **returns * volume** to indicate unusual volume with price move, or **volatility * flows** to see if flows are happening in volatile conditions). The model's gating will handle some of this implicitly, but explicit features might help if certain interactions are consistently predictive.
- **Automated Feature Selection / Reduction:** With ~160 features and possibly more after recommended additions, it might be prudent to use automated feature selection to avoid overfitting. The config mentions options like mutual information or lasso selection [48] . Implementing a **feature importance analysis** using the trained model (e.g. SHAP values, or simply looking at the learned gates from VSN) could identify which features have low importance. Those could be candidates to drop in future retraining. Alternatively, applying a dimensionality reduction like PCA on groups of highly correlated features (like the many moving averages) could reduce feature count and noise. Since the model uses neural networks, one could also consider a learned embedding: e.g. feed technical indicators into an autoencoder or a smaller sub-network to compress them before inputting to the main model. This could let the model learn a lower-dimensional representation of technical state, potentially improving generalization.
- **Data Augmentation:** In time-series, augmentation is tricky, but some techniques can be tried. The **FreqDropout** implemented already acts as an augmentation regularizer in frequency domain [60] . We could also try simple augmentations like adding a bit of noise to input prices or random time warping (shifting a sequence slightly in time). Another idea is **bootstrapping the training data**: e.g., create additional training examples by resampling days (with proper time order) or by mixing parts of sequences from similar days. Care must be taken to not break temporal structure, but a controlled augmentation might improve robustness. For example, one could simulate "what-if" scenarios by randomly dropping one day from the sequence (model should robustly handle missing day) or by perturbing a technical indicator slightly. If the model is sensitive to such small changes, augmentation will help it become more stable. Given the already good performance, augmentation might yield marginal gains, but it can be an insurance against overfitting.
- **Continual Learning and Data Updates:** Ensure the pipeline can easily incorporate new data and **retrain regularly**. In production, new data arrives daily; over months and years, the model should be periodically refreshed to avoid concept drift. The good news is the pipeline is fast and automated, so retraining say weekly or monthly is practical. We suggest setting up a schedule (perhaps via the existing Dagster job scheduler integration [84]) to retrain the model on a rolling basis (e.g., every month retrain using last 3 years of data). This keeps the model parameters in sync with recent market behavior. Also, monitoring of feature distributions and model outputs in production can alert if the model starts performing poorly (e.g., if daily IC drops significantly, that could trigger an earlier retrain or invoking a fallback model).

3. Training Strategy Improvements

- **Direct Optimization of Ranking Metrics:** Since the ultimate goal is high RankIC and Sharpe, one could experiment with incorporating these metrics directly into training. For instance, implement a custom loss term that maximizes RankIC of the predictions in each mini-batch. There are differentiable approximations of rank correlation or methods like differentiable sorting. Even

though these are complex and can be unstable, a small auxiliary loss that nudges the model to produce correct ranking order could improve the final IC. Alternatively, the team might try a two-stage training: first train on quantile loss, then fine-tune the last layer on a loss that directly reflects IC or portfolio return (like a differentiable Sharpe or Sortino ratio). This is an active area of research, so results are not guaranteed, but given the production aim, even a modest boost in IC is valuable. Care must be taken to balance such objectives against the primary loss to avoid overfitting noise.

- **Hyperparameter Tuning & Neural Architecture Search:** The configuration shows an **Optuna** integration for hyperparameter optimization with a search space (learning rate, dropout, layers, hidden units, etc.) [85] [86] . It's wise to fully utilize this to find the best settings. For example, the current hidden size is 256; perhaps a larger model (512) could yield higher accuracy if regularized properly (the search space includes these). Likewise, the number of layers or heads could be tuned. Since training is somewhat costly, one could use a smaller sample of data or fewer epochs to do a relative comparison of configurations, then scale up the best. Additionally, exploring **neural architecture search** for the TFT part (like varying LSTM vs transformer, or adding more skip connections) might uncover better architectures. However, this can get expensive, so a pragmatic approach is to tune the main hyperparams and maybe try a handful of manually designed variants (like "TFT with transformer encoder" or "TFT without GAT") rather than a full NAS.
- **Curriculum Learning and Multi-Stage Training:** If not already fully exploited, the curriculum scheduler could be customized. For instance, start training focusing only on 1-day horizon for a few epochs (horizon weight 1d=1, others=0), then gradually introduce 5d, 10d horizons in the loss. This way, the model first nails short-term patterns then learns to incorporate longer-term info. Another angle is curriculum on feature sets – e.g., train first with technical features only, then add more complex features like flow and options once the model has learned a baseline. This might smooth the learning and prevent being distracted by noisy complex features early on. The current pipeline might not support feature-wise curriculum out-of-the-box, but one can simulate it by freezing certain feature gates initially or gradually increasing regularization.
- **Advanced Regularization:** To further improve generalization, we can add techniques like **Monte Carlo Dropout at inference** (to estimate uncertainty and possibly improve mean predictions), **stochastic weight averaging (SWA)** (averaging model weights over last few epochs, which often yields a more general model). The config already logs multiple checkpoints and can keep top-K models [87] ; using an average of them (SWA) is a quick win. Also, **label smoothing** on the quantile targets (or a small Gaussian noise on training labels) could make the model less sensitive to exact target values, focusing more on rank ordering. Given the model already uses Huber loss, it's somewhat robust, but these minor additions sometimes help in finance where label noise is high.
- **Focus on Cross-Validation Consistency:** In production, one often trains on all available data (sans the last test period). However, a good practice is to validate that the model's performance isn't coming from one lucky split. The team already does multi-fold CV. We recommend taking the **model selected for deployment and testing it on a truly hold-out period** (e.g. the most recent 6 months that were not seen in any fold). This would simulate the next period where the model will be used. If performance holds up, great. If not, consider blending models from different folds or using a rolling update approach. For example, instead of one model, maintain a **model ensemble across time** – models trained on different past windows – and aggregate their predictions. This could hedge against the risk that one training window's model fails if market regime shifts. It's a bit complex operationally, but it can improve robustness. At minimum, doing a "hold-out test" on the latest data before going live is essential to confirm there's no hidden overfitting.

4. Robustness and Operational Considerations

- **Monitoring in Production:** Deploying the model should go hand-in-hand with a monitoring plan. We suggest setting up alerts for if daily RankIC falls below a threshold (the system can compute RankIC on recent predictions vs actuals in a rolling fashion). The existing monitoring dashboard and alert system [64] should be configured with such rules. For example, if a string of days shows negative IC (model doing worse than random), an alert could trigger retraining or fall back to a simpler model until resolved. Also track feature drift – if the distribution of any key feature (say volatility or volume) shifts significantly from the training data distribution (which could happen in extreme market events), the team should be notified as the model might need recalibration.
- **Fallback Strategies:** No model is perfect, so in production it's prudent to have fallbacks. Since a LightGBM baseline model is already available and fast to run [88], it could serve as a backup predictor if for some reason the neural model fails or is not confident. The system could also ensemble the neural prediction with the baseline for a more robust output. The baseline's performance is slightly lower but usually more stable (tree models are less likely to overfit small recent patterns). A simple linear blend (e.g. 80% ATFT prediction, 20% baseline) might improve robustness in unpredictable regimes. This kind of model stacking often yields a small boost or at least protection against the neural model's potential blind spots.
- **Explainability Tools:** In a production setting, especially in finance, explaining model decisions to risk managers or portfolio managers is important. It would be beneficial to integrate explainability: for instance, output the **top contributing features for each prediction**. Since TFT has attention weights and VSN gates, one can extract which features were most influential for a given stock on a given day (the code even stores `feature_gates` and attention weights in the forward pass [9] [89]). Building a small utility to report "This stock was predicted to rise because of X (e.g. strong earnings momentum, high institutional inflows, etc.)" can increase trust in the model. It can also help debug if the model latches onto any spurious signals. Given the complexity, having this interpretability layer will be valuable when communicating with stakeholders or investigating unusual model outputs.
- **Continuous Research and Updates:** Markets evolve, and what works today may decay. A production-level model should be periodically re-evaluated. We recommend setting up a **research feedback loop**: e.g., every quarter, analyze the model's performance, feature importances, and any errors. Look for patterns in where it fails – are there certain sectors or market conditions where predictions are weak? Use that analysis to drive the next iteration of features or model tweaks. The repository already has a strong culture of documentation and iteration (with TODOs, improvement logs, etc.), so continuing this iterative improvement in light of live results will keep performance at peak. For instance, if new types of data become available (say a new sentiment dataset or alternative data), be ready to integrate them as the marginal benefit of new information can be high once the current features saturate.

Finally, **maintain a balance between innovation and reliability**. The current ATFT-GAT-FAN system is at the forefront of AI in finance, and our suggestions aim to push its accuracy higher while shoring up any weak points. By ensembling models, adding carefully chosen data sources, and refining training techniques, the goal of consistently surpassing 0.20 RankIC and achieving robust, real-world trading performance is within reach. With diligent monitoring and periodic updates, the system can adapt to market changes and continue to deliver alpha in production.

Sources:

- Project README and documentation for architecture and improvements [1] [11] [13]
- Data pipeline and feature engineering docs [17] [25] [90] [91]
- Evaluation protocol and training configuration [50] [14] [59]

• ATFT-GAT-FAN code excerpts for model internals [9] [89] [80]

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [65] [66] [67] [79] [80] [81] [89] atft_gat_fan.py
https://github.com/wer-inc/gogooku3/blob/b238fce73fd850e295c79212731e56363e5e95ec/src/atft_gat_fan/models/architectures/atft_gat_fan.py

[11] [12] [13] [58] [60] [62] [64] [73] [74] [77] [82] ATFT_IMPROVEMENTS_TEAM_REVIEW.md
https://github.com/wer-inc/gogooku3/blob/b238fce73fd850e295c79212731e56363e5e95ec/docs/ml/atft/ATFT_IMPROVEMENTS_TEAM_REVIEW.md

[14] [15] [48] [52] [53] [54] [55] [56] [57] [59] [63] [68] [69] [70] [71] [72] [75] [78] [85] [86] [87] [88] unified_config.yaml
https://github.com/wer-inc/gogooku3/blob/b238fce73fd850e295c79212731e56363e5e95ec/configs/atft/unified_config.yaml

[16] [17] [18] [19] [20] [21] [22] [30] [31] [32] [33] [34] [35] [36] [37] [39] [40] [41] [42] [43] [44] [45] [46] [47] [49] [76] [84] [90] [91] data-pipeline.md
https://github.com/wer-inc/gogooku3/blob/b238fce73fd850e295c79212731e56363e5e95ec/docs/architecture/data-pipeline.md

[23] [24] [25] [26] [27] [28] [29] JQUANTS_PHASE2_FEATURES.md
https://github.com/wer-inc/gogooku3/blob/b238fce73fd850e295c79212731e56363e5e95ec/docs/JQUANTS_PHASE2_FEATURES.md

[38] [50] [51] EVALUATION_PROTOCOL.md
https://github.com/wer-inc/gogooku3/blob/b238fce73fd850e295c79212731e56363e5e95ec/docs/EVALUATION_PROTOCOL.md

[61] train_atft_modular.py
https://github.com/wer-inc/gogooku3/blob/b238fce73fd850e295c79212731e56363e5e95ec/scripts/train_atft_modular.py

[83] ablation_report.py
https://github.com/wer-inc/gogooku3/blob/b238fce73fd850e295c79212731e56363e5e95ec/scripts/ablation_report.py