ChatGPT

ATFT-GAT-FAN多ホライズン予測パイプラインのレビューと改善提案

モデル構造の見直しと改善提案

現状のモデル構成: ATFT-GAT-FANモデルは「**Adaptive Temporal Fusion Transformer + Graph Attention + Frequency Adaptive Normalization**」の統合モデルです [1]。実装上は時系列エンコーダとして単層LSTM（**Temporal Fusion Transformer**の簡易版）が使われ、株式間関係の学習に**Graph Attention Network (GAT)**を適用し、正規化層として**周波数適応正規化 (FAN)**と**スライス適応正規化 (SAN)**が導入されています [2]。入力特徴は基本価格・テクニカル指標から派生特徴まで多岐にわたり、静的特徴（銘柄コードのエンコーディングなど）も含まれます [3] [4]。出力は複数分位点の予測で、現状は最終タイムステップの予測分位点5種類（例: 10%, 25%, 50%, 75%, 90%）を吐き出す設定です [5]。

問題点: コードを確認すると、実装はまだ簡易版であり本来意図したアーキテクチャを十分に活かせていません。例えば**TFT**部分はLSTMのみで注意機構や可変選択層などは未実装、**GAT**部分も`edge_index`や`edge_attr`を受け取るものの実際には**全ノード間の単純注意**（PyTorchのMultiheadAttention）になっており、エッジ情報を考慮できていません [6] [7]。**FAN/SAN**もLayerNormに重みパラメータを用意しているだけで、周波数ウィンドウごとの正規化計算は未実装です [8] [9]。その結果、モデルが本来意図した**マルチスケールな時系列パターンの抽出**や**銘柄間関係の動的学習**が十分機能していない可能性があります。

改善提案:

- **Temporal Fusion Transformerの強化:** TFT本来のアーキテクチャ（Gate付きシーケンシャル学習、マルチヘッド注意による短期・長期パターン抽出、可変選択ネットワークなど）を実装します [10]。具体的には、LSTM出力に対して**マルチヘッド注意**を導入し、過去系列と将来系列を統合するTFTのAttentionブロックを追加します。また**可変選択レイヤ**によって特徴量ごとの重要度を学習し、不要な特徴ノイズを低減します [10]。これにより、短期の変動（1〜5日）に効く特徴と中長期のトレンド（10〜20日）に効く特徴をモデルが自動で取捨選択でき、性能向上が期待できます。

- **マルチホライズン出力設計:** 現在は最終時点で分位点のみ出力していますが、**各予測ホライズンごとに専用の出力ヘッド**を持たせることを検討します [11] [12]。例えば`prediction_horizons: [1, 2, 3, 5, 10, 20]`と定義し、それぞれに対応する`PredictionHead`を配置して`point_horizon_1, point_horizon_2, ...`といったキーで出力する構造です [13]。これにより**タスクごとの最適化**が促進され、短期・長期予測の両立がしやすくなります。実装上は`ATFT_GAT_FAN.forward`で最終タイムステップの特徴量から各ホライズンの出力ヘッドを通し、辞書にまとめて返す形に変更します（現状は単一のprediction_head出力なので拡張が必要）。過去の議論でも**各期間に対応する出力ヘッド**の実装が提案されており [11]、実現する価値があります。

- **グラフ注意ネットワークの適切な統合:** GATの効果を高めるには**銘柄間グラフの構築と適用方法**を見直す必要があります。まず、**エッジの定義**をドメイン知識に基づいて改善しましょう。例えば「同一セクターに属する銘柄同士を接続」「過去相関が高い銘柄同士を接続」「親子会社やサプライチェーン関係で接続」等、**経済的な関係性**を反映したグラフを構築します。エッジ属性（`edge_attr`）として相関係数や業種の類似度などを付与し、それをモデルで利用できるよう**注意スコアに組み込む**か、**エッジ重みのプロジェクション**を実装します [14]。実装面では、PyTorch Geometric等のGNNライブラリの`GATConv`を用いて`edge_index`による隣接関係を明示的に考慮した伝播を行う方法が有効で

す。現在の`GraphAttentionNetwork.forward`では全シーケンス次元に対し自己注意を計算していますが[7]、**本来意図する「銘柄ノード間」の注意**にするには、**同時点における各銘柄の埋め込み同士**に注意を適用する必要があります。例えば、「各バッチを単一の観測日（同日全銘柄）に限定し、各銘柄の時系列エンコーダ出力（最終時点の隠れ状態）をノード特徴としてGATを適用する」手法が考えられます。こうすることで、一日の市場に存在する銘柄グラフ上で情報が伝播し、**クロスセクションの関係性**を的確に学習できます[15]。もしバッチ内に複数日が混在している場合には、日次でバッチを分けるか、あるいはattentionマスクを用いて同日以外のノード間注意をゼロにする工夫が必要です。

- **Frequency Adaptive Normalization (FAN)/Slice Adaptive Normalization (SAN)の活用:** 現状の実装ではFAN/SANが単なるLayerNormになっているため[8][9]、これら正規化機構を実際に機能させます。**FAN**のコンセプトは複数の時間的窓で計算した移動統計を用い、それらを学習可能な重みで組み合わせて正規化することにあります。例えばウィンドウサイズ[5, 10, 20]日でそれぞれ特徴量の平均・標準偏差を計算し、短期・中期・長期の変動スケールを把握します。その上で各ウィンドウの統計量を用いて入力を標準化し、`fan_weights`で加重平均するような**適応正規化層**を設けます（設定上は`window_sizes`ごとに学習可能重みがあります[16]）。**SAN**については、系列長をいくつかのスライス（例: 過去20日のうち前半・後半など）に分割し、それぞれで正規化するアプローチが考えられます。例えば過去20日を3スライスに分けて（オーバラップ50%で）特徴量のスケールを調整することで、**時間経過による分布変化**に対応します[17]。これらを適切に実装すれば、短期変動の激しい特徴は短期窓で正規化され、長期的傾向の特徴は長期窓で捉えるといった**周波数帯ごとの特徴強調**が可能になり、予測精度向上が期待できます。

- **静的特徴量と銘柄固有効果の取り込み:** モデル内で**銘柄IDやセクター情報をエンコーディング**し活用できているか確認します。コードでは`n_static_features=10`と仮定して静的特徴をLinear層に通し、LSTM出力に加算しています[18]。この部分を拡張し、**静的特徴をゲーティングに利用**したり、**各銘柄の固有トレンドを補正**する工夫が考えられます。具体的には、各銘柄のIDをEmbeddingベクトルに変換して静的特徴に含め、LSTMの初期隠れ状態を銘柄Embeddingから生成することで、銘柄ごとのパターン（ボラティリティの大きさや平均リターン傾向など）をモデルに持たせます。セクターOne-hotやサイズ（時価総額）といった静的要因もEmbeddingや数値として入力し、**「同じセクターでは似た動きをしやすい」などの事前知識をモデルに提供**します。こうした静的要因は特に中長期予測（10〜20日）の精度に寄与すると考えられます。

以上のようなモデル構造の改善により、短期予測では**直近の勢いと出来高異常**を捉えつつ、中長期予測では**トレンド転換やファンダメンタル要因**を織り込むバランスの良い学習が期待できます。

損失関数と学習戦略の見直し

現状の損失関数: ATFT-GAT-FANモデルでは主損失に**分位点損失（Pinball Loss）**を採用し、オプションで**Sharpe比最大化損失**を加える実装になっています[19][20]。Quantile Lossは将来リターンの分布予測に対応するためですが、短期〜長期の複数ホライズンを一括で最適化するには工夫が必要です。また、損失関数がモデルの学習目標となるため、**どの指標を重視するか**を明確に反映させることが重要です。

改善提案:

- **多ホライズン損失の導入と重み付け調整:** 複数ホライズンの予測精度をバランス良く向上させるため、**Multi-Horizon Loss**を明示的に導入します。既存コードにも多ホライズン損失のクラス（Huberベース + Horizon重み + 分位点）が用意されています[21][22]。これを活用し、**短期ホライズンに高い重み**を割り当てつつ長期ホライズンも無視しないようにします。デフォルトでは例: 1日=1.0, 2日=0.8, 3日=0.7, 5日=0.5, 10日=0.3 といった重みになっています[22]。現状短期・中期の性能が伸び悩んでいるとのことですので、例えば**ホライズン1〜3日の重みをさらに高め**（1: 1.0→1.2 など）つつ、ホライズン

10日や20日も一定の重み（0.1〜0.2程度）を持たせて学習から外れないようにします [23]。モデル設定でまだ20日ホライズンが含まれていない場合は、**予測ホライズンに20日も加えて**損失計算・評価するよう変更すべきです（Baselineでは1/5/10/20日全て予測対象でした [24]）。

- **ロバスト損失関数の採用:** リターンは外れ値が多いため、**Huber損失**や**Quantile損失**の採用は妥当ですが、さらなるロバスト性向上のため**Student-t分布の対数尤度損失**を組み込むことも検討します。実際、環境変数`USE_T_NLL=1`により**Student-t分布NLL**を有効化する設定が用意されており [25] [26]、Sharpe 0.849を達成した際にはPinball LossとStudent-t NLLのハイブリッド損失が使われました [23]。Student-tは裾の重い分布を仮定するため外れ値に寛容であり、**リスクの大きい局面でも極端な損失勾配が発生しにくい**利点があります。具体的には、学習初期はPinball（分位点）損失を主に用い、モデルが安定してきた数エポック後に`USE_T_NLL`を1にしてStudent-t NLLを併用/置換するような**段階的損失切替**戦略が提案されています [27] [28]。これにより、まず中央値等の点予測を安定させてから分布の裾野（リスク部分）までモデルが表現できるようになります。

- **評価指標駆動型の補助損失:** 最終的に重視したい指標（例えばSharpeやRankIC）がある場合、それを損失関数にも反映させます。既に**Sharpe比最大化損失**（負のSharpeを与える）や**RankIC最大化損失**の実装があり [29] [30]、環境変数で有効化が制御されています [31]。短期リターンの方向性を当てることが重要であれば**RankIC損失**を、利益率自体を最大化したいなら**Sharpe損失**をそれぞれ適度な重みで加えるのが有効です。例えばSharpe損失に0.05、RankIC損失に0.1といった比率で組み込み [32]、メインの多ホライズン損失にこれらを加算します。これによりモデルは**「予測と実際の順位相関を高めつつ、リスク調整利益も意識する」**ようになります。ただし重みが大きすぎると勾配が暴れ学習が不安定化するため、小さめから調整し、**学習が進むにつれて徐々に重みを上げる**（例えばエポックが進むごとにSharpe損失重みを0→0.05へ線形に増やす）といった工夫も考えられます。

- **方向性精度向上の工夫: 勝率（Up/Downの的中率）**を直接向上させたい場合、損失関数に**分類タスク的な要素**を入れることも可能です。例えばターゲットの符号（上昇=1/下降=0）を予測するサブヘッドを設け、**バイナリクロスエントロピー損失**を追加で学習させます。ただ、これは回帰タスクと並行するため調整が難しく、RankICやSharpeの最適化でも方向性はある程度改善されることが多いです。そのためまずは前述のRankIC損失導入で様子を見て、必要なら分類ヘッドを追加するとよいでしょう。

- **ラベルのクリッピング:** 学習安定化のため、**ターゲットとなるリターン値にクリッピングを施す**のも有効です。特に株式リターンは異常値（サーキットブレーカー級の変動など）があり、それらがあるとMSEやQuantile Lossが過大なペナルティを生み学習を妨げます。環境設定では`LABEL_CLIP_BPS_MAP="1:2000,2:2000,3:2000,5:2000,10:5000"`のように各ホライズンのリターンをbps単位でクリップする指定がされていました [33] [34]。これは例えば**1日リターンは±20%、10日リターンは±50%**に外れ値を制限することを意味します。実運用上もそれ以上の変動は稀かつ予測困難なので、事前にターゲットをこの範囲に収めることで損失計算が極端な値に引っ張られず、安定した勾配で学習できます。実装上はデータローダでターゲットを読み込む際に`clip`処理を入れるか、学習時に`torch.clamp`で制限します。

- **損失関数の段階的適用（Phased Training）:** 複雑な損失を一度に最適化しようとすると不安定になる場合、**学習過程で段階的に目的を増やす**戦略が有効です。例えば:

- **Phase1:** 学習初期（例: 2エポック）は**多ホライズンHuber損失**のみに集中し、まず基本的な予測精度を確保する [35]。
- **Phase2:** 次の段階で**RankIC損失やSharpe損失を有効化**し、モデルにランキング精度やリスク意識を持たせる [32]。
- **Phase3:** 学習後期に**Student-t NLLやQuantile Loss**を組み入れ、予測分布の形状や不確実性の表現を仕上げる [26]。

こうした段階は、コード中でエポック数に応じてフラグを切り替えるか、学習をチェックポイント区切りで再開して環境変数を変更することで実現できます [27] [28]。段階的学習により、まずモデルにとって難易度の低い目標で土台を作り、その上で徐々に高次の目標にシフトしていくことができます。これにより**学習の安定性**と**指標最適化**を両立させ、結果的に短期・長期両面の性能向上が期待できます。

以上のような損失関数設計の見直しにより、モデルが**「何を重視すべきか」**を明確に示し、短期リターンの精度と中長期リターンの有用性（ランキング相関や実運用での利益指標）を同時に向上させることができます。

入力設計と系列長の調整

現状の系列長と特徴設計:　モデルは過去**20営業日**程度の系列データを入力としているようです（設定で `max_sequence_length: 20` [36]）。特徴量は価格・テクニカル・移動平均・フロー・リターンなど総計300次元以上に及ぶ多彩なものが投入されています [37] [4]。この豊富な特徴量が適切に機能していれば良いですが、系列長や入力の持たせ方によっては**短期パターンを捉えきれない**、**長期予測に必要な情報が不足**といった可能性があります。

改善提案:

- **系列長の見直し:**　**短期予測(1〜5日)**では直近数日の値動きが主要因となる一方、**長期予測(10〜20日)**では1か月程度のトレンド把握が重要です。現状20日間の履歴では長期予測にはやや短い可能性があります。系列長を例えば**60営業日（約3か月）**程度まで延長し、**長めのトレンドやサイクル**も捉えられるようにすることを検討してください。実装上はデータローダ側でシーケンス長を変更するか、設定 `data.sequence_length` を更新します（例: `sequence_length: 60`）。系列を長くすると計算負荷は増えますが、**長期予測の土台となる情報量**が増えるメリットがあります。逆に短期予測性能への悪影響が懸念されますが、モデルが不必要な過去情報を無視できるよう前述の可変選択機構や注意機構を組み合わせれば、長めの系列の中から短期予測に本質的な直近期の情報だけを抽出できます。

- **マルチスケール入力の導入:** 単一のLSTMに長い系列全てを入れるのでなく、**異なる期間の特徴を別経路で扱う**工夫も有効です。例えば**Dual Encoder**のように、直近5日の微細な動きを捉える短期エンコーダと、過去60日の大局的トレンドを捉える長期エンコーダを用意し、それぞれの出力を統合して予測に活かすアーキテクチャが考えられます。具体的には、5日程度の短系列を入力にとるCNNやLSTMで短期変動成分を抽出し、60日程度の長系列を入力に別のLSTMで長期傾向成分を抽出、最後に結合してPredictionHeadに渡す形です。これにより**短期・長期のパターンを並列に学習**でき、両者の情報を漏らさず捉えることができます。このような改良は大掛かりですが、多ホライズン予測の性能ボトルネックを突破する一手です。

- **特徴量の有用性検証と選抜:**　入力特徴が非常に多岐にわたるため、モデルが**重要なシグナルにフォーカスできていない**可能性があります。Baselineモデルでは特徴量重要度の分析も行われています [24]。同様に、**各特徴の寄与度（ゲインやSHAP値）**を算出し、あまり効果のない特徴は思い切って削減することも視野に入れます。特に短期予測に効く特徴（例:　短期モメンタム指標や出来高急増フラグ）と長期予測に効く特徴（例:　長期移動平均乖離やボリンジャーバンドの幅）を見極め、モデル入力を整理することが重要です。また、**情報漏洩に注意**して特徴設計を確認してください。例えば「リターン5日」列をホライズン5日のターゲットと別に持っている場合、適切にシフトされていないと未来情報の漏洩になります。現状のデータロードで `_normalize_target_key` 関数により様々なカラム名を正規化してhorizon_{h}に対応付けているので [38] [39]、ラグの扱いは正しいと想定されますが、**特徴量作成時に将来値を参照していないか**改めて点検しましょう（例えば移動平均やテクニカル指標が直近値までで計算されているか、ターゲット期間を含んでいないかなど）。

- **欠損値・初期期間のマスキング:** 銘柄ごとに上場日が異なるため、データセット冒頭の**履歴が足りない部分**や、指標計算で発生する**NaN**は適切にマスクまたは埋められているか確認します。Baselineでは特徴・ターゲットのNaNを0埋めしています [40] が、モデル入力として0が大量にあると特殊な信号として扱われかねません。より良い方法は**マスキング**です。PyTorchのPackedSequenceを使えば可変長シーケンスを直接扱えますし、あるいは**無効なタイムステップでは損失を計算しない**工夫もできます。例えば、新規上場銘柄の最初のN日間は `valid_flag=0` のようなフラグを入力特徴に含め、モデルがその期間のデータを無視するよう学習させる方法があります（実際、特徴量に各指標の有効フラグが含まれています [3] ）。**初期パディング部分を明示的にゼロマスク**する実装をLSTMやAttentionに組み込めば、学習が安定し精度も向上します。短期予測では特に**前日データが存在しない場合は予測不能**なので、そのようなケースは損失計算から除外することを徹底します。

以上のように、入力データ自体の改善を行うことでモデルが**必要な情報を十分に受け取り、不要な情報やノイズに惑わされない**ようにできます。系列長と特徴量の適切な設計は、モデル性能の土台を形作る重要ポイントです。

特徴量の正規化とクロスセクション・マスキング処理

現状の正規化: モデル内部では入力投影直後にLayerNormを入れる設定が可能になっています [41] が、データ全体のスケーリングやクロスセクション正規化については明示されていません。Baselineでは**日次のクロスセクション正規化**が使われています（ `CrossSectionalNormalizerV2` ） [42] [43] 。これは各日について全銘柄の特徴量を標準化する処理で、市場横断的な比較をしやすくする効果があります。

改善提案:

- **クロスセクション正規化の導入:** Deep LearningモデルでもBaselineに倣い**日単位の標準化**を取り入れることを検討してください。具体的にはデータローダで同一日付の全銘柄について各特徴量の分布（平均・標準偏差）を計算し、その日内でZスコア正規化する方法です。こうすることで、例えば「出来高が1億の銘柄」と「100万の銘柄」の規模差を取り除き、**その日内で相対的に大きい/小さい値か**という情報だけをモデルに与えることができます。Baselineの `FinancialMetrics` 実装にも「**評価指標は日次クロスセクション単位で計算すべき**」とあり [15] 、モデル入力も同様に**日ごとの横断比較ができるスケール**に揃えるのが望ましいです。実装としては、データ読み込み時にPandas/Polarsで日付ごとにグループ化し `(X - mean(X_day))/std(X_day)` を適用するか、あるいはDataLoader内でバッチ＝1日分としバッチ正規化する方法があります。PyTorch LightningのDataModuleを使っているなら、DataModule内で正規化処理を一括で行うと安全です。

- **ターゲットの標準化・スケーリング:** 特徴量だけでなく**ターゲットのスケーリング**も検討します。Sharpe比などリスク調整指標を見る際はターゲットをそのままのスケールで扱いますが、学習を安定させる目的ではターゲットリターンを例えば**対数リターン**に変換したり、標準偏差で割って**単位リスク当たりのリターン**にスケールする方法もあります。もっとも、本プロジェクトではターゲットは既にbps換算されているか、Sharpe損失などで標準化効果が入っている可能性があるため、二重に標準化しないよう注意します。基本的には**特徴量側は正規化、ターゲット側は必要ならクリップ**程度に留め、損失関数内でSharpeやNLLを使うことでリスクリターン比を学習させる方針が良いでしょう。

- **Batch Normalizationなどの活用:** 入力特徴が多くスケールも様々であれば、モデル内部で**BatchNorm**や**LayerNorm**を適所に入れるのも有効です。すでにInput Projection後にLayerNormを入れるオプションがありますが [44] 、これを有効化する（ `config.model.input_projection.use_layer_norm=true` [45] ）のは基本です。さらに、PredictionHead内の隠れ層で `use_batch_norm` 設定もあります [46] がデフォルトfalseなので、もし学習が不安定ならtrueにしてミニバッチ単位の正規化を試す価値があります。BatchNormは時系列データには直接は合わない場合もありますが、隠れ層の分布変動を抑えるには寄与します。ただし**時系列**

モデルではBatchNormがデータの時間的相関を壊す恐れもあるため、基本はLayerNormやGroupNormなど系列依存しないノーマライゼーションが無難です。FAN/SANがうまく動けばそれが最適な正規化になります。

- **マスキング処理の実装:** 前述したように、**無効なデータポイントのマスク**はデータロード段階とモデル計算段階の双方で検討します。DataLoaderでシーケンスを整形する際、各シーケンスの長さや有効フラグを保持し、LSTMに`pack_padded_sequence`を使って与えると自動的にマスクされます。またAttention機構を使う場合はマスク行列を用意して、無効部分にはアテンション重みがかからないようにできます。これら実装には手間がかかりますが、**パディング部分や将来情報への注意を遮断**することは予測の公正性のため必須です。例えば過去に**銘柄の上場前期間が0埋めされており、その0にモデルがパターンを見出してしまった**ケースがないか確認してください。マスク実装によってそうした問題は原理的に防げます。

以上、正規化とマスキングの適切な適用により、モデルは**入力スケールのばらつきや無効データに煩わされず**に学習でき、特にクロスセクション正規化は**全銘柄を横断した予測能力**に直結するため、中長期のRankIC向上に効いてきます。

グラフ構築手法の見直し

現状のグラフ構築: Graph Attention Network部分で使われるグラフ（edge_index, edge_attr）は、実装を見る限り**静的な設定**かつ**完全グラフに近い扱い**になっているようです [7]。エッジ特徴量次元が3と設定されていることから [14]、何らかの3種の関係（例: 相関, 業種同一, サプライチェーン?）を考慮している可能性がありますが、詳細は不明です。性能が伸び悩んでいる一因として、**グラフの構築方法や使い方が不適切**なことが考えられます。

改善提案:

- **経済的な関係に基づくエッジ定義:** グラフのエッジはドメイン知識を反映して選択することが重要です。例えば以下のような手法が考えられます:
- **業種・セクターエッジ:** 同じ業種（セクター）内の銘柄同士は強い共通要因を持つためエッジで接続する。エッジ重み/属性として業種の同一性（1 or 0）やサブセクター間距離を与える。
- **相関エッジ:** 過去○ヶ月の株価リターン相関が高いペア同士にエッジを張る。例えば相関係数上位5位以内の銘柄同士を結ぶか、一定閾値（例: 相関>0.5）以上で結ぶ。また相関値自体をedge_attrとして付与する [14]。
- **地理・指数連動エッジ:** 同じ国・指数に属する銘柄同士（TOPIXコア30などグループ）を接続する。
- **その他関係:** 親子上場企業、サプライチェーン（自動車メーカーと部品会社）など特定ペアをドメイン知識で繋ぐ。

これらを**重み付き多重グラフ**として統合し、edge_attrの次元を増やしても良いでしょう。現在edge_attrが3次元なら、「業種同一フラグ」「相関値」「相関有意フラグ」のようなものかもしれませんが、要件に合わせて見直します。

- **動的グラフへの拡張:** 株式間の関係性は時間とともに変化します。例えばコロナショック時には同業種内の相関が一時的に跳ね上がる等があります。そこで**時間とともに変化するグラフ**をモデルに取り入れることも検討します。具体的には、**スライディングウィンドウで相関を計算し直してエッジ更新**したり、四半期ごとにセクター分類や時価総額の変動を反映するなどです。実装上は、DataLoaderが日付ごとに異なるedge_index/edge_attrを提供できるようにする必要があります。PyTorch GeometricのTemporalGraphConvなども視野に入りますが、シンプルには**「各日について独立したグラフを設定し、その日のバッチではそのグラフを使う」**形が取りやすいです。Baselineのウォークフォワード検証を応用し、検証期間ごとに相関行列からエッジ構造を組み直すアプローチもあります。

- **グラフ適用箇所の見直し:** 先述の通り、**グラフは各日での銘柄間関係に適用**すべきです。現在の実装では系列長方向に注意をかけているだけなので、これを**バッチ内（銘柄間）**にかけるよう修正します。もしPyTorch　Lightningのミニバッチ分けで日が混在するなら、バッチサイズ=1日（全銘柄）に設定し直し、1ステップで全銘柄の予測を行うようにする必要があります [15]。大量の銘柄を同時処理するコストは上がりますが、A100　80GB等の環境なら4000銘柄程度一括処理も可能との前提で、**日次でGraph層を適用→その日全銘柄の予測出力**という設計にすると、GATが本来意図する効果を発揮できます。

- **GAT融合アルファの調整:**　環境変数で`GAT_ALPHA_INIT`や`GAT_ALPHA_MIN`が定義されているように、**TFT出力とGAT出力を線形結合する重みα**が想定されています [47]。これは、学習初期はα=0.5などとしてTFTとGATを半々で使い、学習が進むとαを減衰させてGATの影響力を制限するといった**融合制御**の目的があります。実際Sharpe 0.849の再現設定では`GAT_ALPHA_INIT=0.3, GAT_ALPHA_MIN=0.1`が用いられていました [48]。これに倣い、**モデル実装でもαによる出力ブレンド**を導入しましょう。例えば`combined_features = α * tft_output + (1-α) * graph_output`とし、αを学習中はハイパーパラメータとして徐々に減らすか、あるいは損失に`α`に対するペナルティ項（L2正則化的に(α-目標値)^2）を加える方法があります [47]。後者の場合、`GAT_ALPHA_PENALTY`を環境変数で与え、目標のα最小値（0.1）まで下げるペナルティを課す実装が想定されます [49]。これにより、学習序盤はグラフも使って多様な関係を探索しつつ、終盤は必要最小限のグラフ効果に絞って過適合を防ぐ、といった**段階的グラフ統合**が可能です。

- **Attentionのスパース化:** 全銘柄間の注意は計算量・学習安定性の面で負荷が高くなります。エッジが疎なグラフでも、現在の実装は全ノードに対する完全グラフ的Attentionになっているため、**不要な注意をカットする**仕組みを入れると良いでしょう。例えば**隣接行列に対応したマスク**をMultiheadAttentionに渡し、エッジで繋がっていない組はスコア0になるようにします。PyTorchの`nn.MultiheadAttention`は`attn_mask`引数で接続を遮断できますので、edge_indexからマスク行列を生成して適用可能です。ただ、この自前実装は手間がかかるため、より簡潔には**PyGのGATConv**等を使って隣接行列を直接処理した方が安全です。加えて、環境変数`SPARSITY_LAMBDA`にあるように**注意行列のスパースネスにペナルティ**を課すことも検討されています [49]。これはAttentionのエントロピーに罰則を与え均等な注意よりも尖った注意を促すテクニックです（重要な関係に絞る）。実装上はAttentionのsoftmax出力に対し$\sum　p　\log　p$のようなエントロピー計算を行い、それにλを掛けて損失に加える方法になります。現状`gat.layer_config.attention_entropy_penalty=0.001`と設定があるので [50]、これをGATのフォワード内で適用するよう実装（PyTorchでは`attn_output, attn_weights = self.attention(...)`から`attn_weights`のエントロピーを計算）します。こうした調整により、**モデルはより意味のある銘柄関係に集中**でき、ノイズ的な関係に惑わされにくくなります。

以上、グラフ構築・活用法を見直すことで、銘柄間関係から得られる情報を**モデルが正しく捉えられる環境**を整えます。特に短期予測ではマーケット全体の急変動（システムリスク）は銘柄間で伝播しやすく、グラフを通じてそのシグナルを共有することで**方向性の的中率向上**に繋がるでしょう。また長期予測ではセクター循環や構造的な連動を掴む助けとなり、**RankICやSharpeの改善**が期待できます。

トレーニングハイパーパラメータの調整

現状のハイパーパラメータ: 提供されたスクリプトでは、例として**学習率5e-5**, **バッチサイズ1024**, **エポック数50**, **早期終了patience=10**等が使用されていました [51]。OptimizerはAdamWでWeight Decay 1e-4、学習率スケジューラはCosineAnnealingWarmRestartsが既定になっています [52] [53]。これら設定がモデルに最適か、再検討の余地があります。

改善提案:

- **バッチサイズと学習率のバランス:** バッチサイズ1024は大きめで、損失表面をなだらかにし学習安定には寄与しますが、勾配の多様性が減り**局所解にハマりやすい**側面もあります。実際、Sharpe 0.849を達成した設定では**バッチサイズ256**が用いられており [34]、大きすぎないバッチで勾配ノイズを確保する戦略が取られました。そこで**バッチサイズを例えば256〜512程度**に下げることを提案します。ただしその場合1エポックあたりのイテレーション数は増えるため、学習率を調整する必要があります（一般にバッチ4倍ならLRも2倍など）。現在5e-5という比較的低LRですが、バッチ縮小に伴い**LRを1e-4程度に上げる**ことも試してみてください。逆に、モデルが不安定な場合はバッチを小さくしてLRも小さく（例: 3e-5）する選択肢もあります。適切な組み合わせを見つけるため、小規模実験で**学習曲線の安定性**と**指標の推移**を観察してください。

- **エポック数とEarly Stopping:** 現在50エポック・patience 10となっていますが、**学習が収束しきっていない**可能性があります。特に分位点や複合損失を扱う場合、損失減少が遅く長めに学習した方が良いケースもあります。Sharpe重視なら損失最小よりも**指標最大化のタイミング**で止めるべき場合もあるため、**EarlyStoppingのモニタ指標**を再考します。例えば短期の方向性を重視するなら`val_rankic`や`val_hit_rate`を監視し、一定期間改善しなければ終了という方法もあります。ただ指標はノイズが大きいので、まずはエポック数を増やして十分学習させ、その上で過学習兆候（Val lossの上昇等）を見て判断します。A100 80GB環境なら多少エポックを伸ばしても時間的余裕はあると思われますので、**エポック100前後まで**試すことも検討してください。

- **学習率スケジューラとWarmup:** 現状CosineAnnealingWarmRestartsで周期的にLRを復活させる戦略ですが、もし学習初期の不安定さが問題であれば**Linear Warmup + Cosine Decay**のようなスケジューラも効果的です。例えば最初の1エポックでLRを徐々に5e-5まで上げ、その後CosineAnnealingで減衰させると、勾配爆発のリスクが下がります。LightningではOneCycleLRやLinearWarmupの組み合わせも簡単に導入できます。環境変数にも`ForceMode`による段階学習がありましたが [27]、それと組み合わせ、**Phase2移行時にLRをリセットして再Warmup**するなど高度な調整も可能です。まずは単純な**ReduceLROnPlateau**で指標が伸び悩んだらLRを半減、といった戦略も検討できます。

- **混合精度と勾配安定化:** 環境設定では`USE_AMP=1`（混合精度 BF16）や`gradient_clip_val=1.0`が推奨されています [54] [55]。これらは既に使っているかもしれませんが、もし未設定なら**有効化**してください。A100であればBF16混合精度は計算高速化とGPUメモリ節約になり、学習も安定しやすい傾向があります。また**勾配クリッピング**は必須と言えます（現在Norm1.0でクリップする設定 [55] は適切）。勾配爆発の兆候があればさらに閾値を下げる（0.5など）ことも検討します。

- **学習安定化のためのテクニック:** Sharpe 0.849達成時には**崩壊防止や予測分散維持**の工夫が多々盛り込まれていました。例えば:

- **崩壊（degeneracy）ガード:** 出力が単一値に潰れる現象を検知・防止する仕組みです。`DEGENERACY_GUARD=1`で有効化され、一定ウォームアップ以降で予測分散が極端に小さい場合に警告/ペナルティを発するようになっています [56]。具体的には、バッチ内予測の標準偏差が`DEGENERACY_MIN_RATIO`未満になると発動するようなチェックです。これをLightningのトレーニングループに組み込み、もし発生したら学習率を下げる・訓練中断するなどの対応ができるようにします [57]。少なくとも学習ログにその値を出力し、注意深くモニタリングしてください。

- **予測分散ペナルティ:** 環境変数`PRED_VAR_MIN=0.01`, `PRED_VAR_WEIGHT=1.0`などは**予測の多様性を強制する**目的です [58]。具体的には、モデルの出力（例えば全銘柄予測）について分散が一定以下ならペナルティ損失を加える仕組みです。これも極端な自信過剰なモデル（全銘柄同じ予測を出すような状態）を避けるのに有効です。既存コードに該当の実装がなければ、自前で`predictions.var(dim=?)`を計算し閾値以下なら損失+=λを加える処理を追加可能です。もっと単純には**Dropoutを適切に効かせる**ことで予測分散は確保しやすくなります。現状InputProjectionや

PredictionHeadでDropoutを入れていますが [44] [59]、必要に応じてドロップ率を0.2→0.3など上げることも検討してください。

- **ヘッドノイズの注入:** 学習初期に**出力層に小さなノイズを加える**と局所解からの脱出に有効です [60]。環境では`HEAD_NOISE_STD=0.02`をウォームアップ2エポック適用し、その後オフにする設定でした [61] [35]。実装は簡単で、PredictionHeadのforwardで`output += torch.randn_like(output)*σ`を加えるだけです（学習時のみ適用）。このノイズにより、最初のうちはモデルがあまりにも自信のある予測（極端な値）を出しにくくなり、安定してloss地形を探索できます。その後十分学習したらノイズを消すことで精度を損ないません。`OUTPUT_NOISE_STD`も似た目的で、これは全出力に対し一定ノイズを掛けるものですが、まずはヘッドノイズだけでも導入してみてください [62]。

- **段階的学習戦略再確認:** 既に述べた**Phase毎の設定変更**をハイパーパラメータ観点で補足します。Phase1（TFTのみ）では`FUSE_FORCE_MODE=tft_only`でGATを切り離し [35]、Phase2移行時にこれを解除し`EDGE_DROPOUT_INPUT_P=0.1`でグラフ正則化を開始します [63]。最終Phaseで`HEAD_NOISE_STD=0`にしてノイズ除去、`USE_T_NLL=1`で分布予測をオンにする [64]、といった切替を行います。これらは**エポック数や時間でトリガー**できます。Lightningではコールバックで現在エポックを見て`os.environ`や`model.config`を更新することも可能です。多少運用は複雑になりますが、その分モデルの性能を引き出せる戦略です。

- **ハイパーパラメータチューニング:** 最後に、以上の要素には未調整のハイパーパラメータが多く含まれます。**体系的なハイパーチューニング**（Grid SearchやBayesian Optimization）も視野に入れましょう。例えば:

  - 学習率5e-5 vs 1e-4 vs 5e-4
  - バッチサイズ256 vs 512
  - Horizon重み（1日:0.6,0.8,1.0 等）
  - Sharpe損失重み0.01〜0.1, RankIC重み0.1〜0.5
  - ドロップアウト率0.1〜0.3
  - LSTM層数1 vs 2、隠れサイズ64 vs 128
  - GATヘッド数やレイヤ数（現状heads=[4,2], num_layers=2 [65] だが、1層の方が安定なら変更）

などです。特に**短期と長期のトレードオフ**になるパラメータ（ホライズン重みやRankIC重み）は、目標に応じて慎重に探る必要があります。一度に多くを調整すると大変ですが、まず**損失関数関連**→次に**学習率・バッチ**→**モデル容量**の順で感度分析すると効率的です。W&Bなどを使ってスイープ実験を行えば、自動でSharpeやICの高い領域を提案してくれるでしょう。

以上、ハイパーパラメータの観点から学習設定を最適化することで、モデル本来の性能を引き出しやすくなります。特に**学習の安定化策**（ノイズ注入や崩壊防止）は短期リターンの鋭敏なシグナルを損なわずにモデルを育てるのに寄与し、**適切なLR/バッチ設定**は中長期トレンドを過学習せず捉える助けとなるでしょう。

評価指標の選定と運用提案

評価指標の重要性: モデル性能を適切に評価し、非専門家でも理解しやすい形で示すには、**金融実務に直結した指標**や**直感的な精度指標**を組み合わせることが重要です。ご提示のあったSharpe・方向勝率・RankIC・MSEは、それぞれ異なる側面を表す指標であり、総合的にモデルの良し悪しを判断する助けになります。以下、それぞれの指標の意味と活用法、そして実装方法の提案です。

- **Sharpe比（リスク調整リターン）:** モデルの予測に基づいて投資戦略を構築した場合の**リスクあたりリターン**を示す指標です。非専門家にも「**1以上なら優秀**（1年あたりリスク1に対しリターン1以上）」程度の目安で説明できます。Sharpe比は**(平均リターン) / (リターンの標準偏差) * √252**で年率

換算します [66] 。モデル評価でSharpeを使うには、**モデル予測をどうポートフォリオに反映するか**を定義する必要があります。一般的な方法:

- **方向戦略:** モデルが予測した上昇確率または予測リターンが正の銘柄を買い、負の銘柄を空売り（または何もしない）すると仮定します。等金額で持つと1日あたりのポートフォリオ実現リターンがモデルの「方向的中度合い」に比例します。この等ウェイトロングショート戦略のリターン系列を日次で作り、そのSharpe比を計算します。
- **スコアに比例したポートフォリオ:** モデルの予測値そのもの（例えば5日後リターンの予想）を各銘柄のポジションサイズとみなし、予測に比例してウェイトを取る戦略を考えます。実現リターン＝Σ(予測スコア * 実現リターン)となるので、モデルの予測と実績の共分散を反映します。この戦略のSharpeは情報比(IC)とも関連します。

Sharpeは**金額やリスク制約を含めた実運用適性**を見るのに優れており、中長期のモデル比較では特に重要です。「Sharpeが高い＝リスクの割にリターンが良い予測」と理解できます。実装面では、Validationデータについて**日次でモデルの予測に沿った仮想ポートフォリオを組んだ場合の損益**を計算し、それからSharpeを算出します。簡易には、ATFTモデル内で**各検証バッチ（1日分）ごと**に平均リターンを算出し [66] 、累積積算したポートフォリオ曲線からSharpeを計算することも可能です [67] 。ただしバッチ単位ではなく全期間通しで計算するには、検証完了後にまとめて計算する必要があります。BaselineのFinancialMetricsにはポートフォリオのdecile分析機能もあります [68] 。余裕があれば**トップ銘柄を持つ戦略のSharpe**や**スコア順Decileポートフォリオの**Sharpeなど詳細分析も行うと良いでしょう。

- **方向勝率（Hit Rate）:** 予測した方向（上昇/下降）の的中率です [69] 。これは**直感的な精度指標**として非常にわかりやすく、非専門家には「**当たり/ハズレの割合**」として伝えられます。例えば「このモデルは明日の株価が上がるか下がるかを60%の確率で当てます」と言えれば一目瞭然です。勝率50%がランダム相当で、それを上回っていれば有用性があると判断できます。短期予測では特にこの勝率が意思決定に直結します（売買判断の成功率として）。実装は単純で、**予測値と実際のリターンの符号を比較**するだけです [69] 。ATFTモデルでは検証時に `predictions > 0` と `targets > 0` を比較して平均を取ることで計算しています [69] 。これはホライズンごとにも計算可能で、例えばhorizon_1の勝率、horizon_5の勝率と出せば、短期と中期でモデルの当たりやすさがどう違うか評価できます。勝率は単独では不十分で、例えば勝率51%でも損大利小では負けてしまうのでSharpe等と併せて解釈すべきですが、「**予測の方向精度**」という観点でチーム内に説明するには適切な指標です。

- **RankIC（順位相関係数）:** モデルが**将来リターンの大小関係をどれだけ正しく並べられているか**を示す指標です。Spearmanランク相関で計算され、+1が完全一致、0がランダム、-1が逆相関となります。金融では**Information Coefficient (IC)**とも呼ばれ、日次のRankICを平均したものは長期のモデル有効性を表します [70] [71] 。RankICは**ポートフォリオ組成**に直結する指標です。例えばRankICが高ければ、モデル予測に従い銘柄をランキングして上位を買い下位を売る戦略が有効であることを意味します。非専門家には若干なじみが薄いかもしれませんが、「**1日ごとに見て、モデルの予想ランキングと実際の騰落ランキングに相関があるか**」と説明できます。一般に0.1程度でも有意と言われますが、分かりやすさのため**%勝率に言い換える**こともあります（例: RankIC 0.1相当は上位10銘柄中5〜6銘柄は当たるイメージ等）。実務では**0以上を安定して出せれば収益化可能**とされるため、Sharpeなどと併せてモデルの有効性を示す根拠になります。実装はBaselineにある通り、**日ごとにSpearman相関を計算し平均**します [72] [71] 。FinancialMetricsクラスの `compute_rank_ic` を利用すれば、予測値配列と実績リターン配列と日付配列を入れるだけで計算できます [73] 。ATFTモデル内で逐次計算するより、検証終了後に全検証データで一括計算する方が正確です。例えばLightningの `on_validation_epoch_end` で全バッチの予測・実績を蓄積し、epoch終わりにRankICを計算して `logger.log_metrics` する、といった方法が考えられます。RankICは特に**中長期予測の評価**に重要で、たとえ絶対的な誤差(MSE)が大きくてもRankICが高ければモデルには投資価値があると判断できます。

- **MSE（Mean Squared Error）/RMSE:** これは**純粋な数値予測精度**を表す指標です。モデル予測値と実際のリターン値の平均二乗誤差で、スケールに依存しますがモデル全体の**当て勘**を見るには基本と

なります。非専門家には「**予測と実績のズレの大きさ**」として説明できます。例えば「RMSE=0.02（=2%）なら、平均的に予測は実際のリターンと±2%程度ズレます」という具合です。MSE/MAEは機械学習では標準指標ですが、金融ではそれ自体が利益に直結しないためSharpeやICほど重視されません。しかし**モデル比較**には有用で、MSEが小さいモデルは概ね他の指標も良くなる傾向があります。特に**短期ホライズン**では勝率50%を超えればMSEもある程度小さくなるので、参考値として追跡すると良いでしょう。実装は言うまでもなく、検証データでの`(prediction - actual)^2`の平均です。Lightningでは`val_loss`としてQuantile LossやHuberを使っていますが、**検証ループ内で別途MSEを計算してログ**することも可能です（PyTorchのF.mse_lossを使うか、自前計算）。MSEは数値の解釈が難しい場合もあるので、**ベンチマークとの相対比較**が大事です。例えば「ベンチマークモデルのMSEを1としたとき本モデルは0.9（=10%誤差減少）」のように伝えると分かりやすくなります。

総合的な指標選定: 上記の指標はそれぞれ意味が異なるため、**組み合わせて評価・レポート**することを推奨します。具体的には: - **短期予測評価:** 勝率とRMSEを重視し、補助的にRankICも見る。勝率が50%を明確に超え、RMSEが過去モデルより低ければ短期精度向上と言えます。 - **中長期予測評価:** RankICとSharpe比を重視する。RankICがプラスで高水準、Sharpe比も1以上であれば長期でも有効なモデルと判断します。 - **全体評価:** 上記指標をテーブル化し、例えば「Sharpe0.8, 勝率55%, RankIC0.05, RMSE0.015」といった形で提示します。非専門家にはSharpeと勝率をまず示し、「Sharpe0.8で勝率55%です」と伝え、詳しい人にはRankICやRMSEの値も共有する形が考えられます。

コードへの組み込み: 既にATFTモデルは検証時にSharpe比・最大ドローダウン・勝率を計算しログに出す処理があります [6] 。これを拡張・修正して以下を行います: - **Sharpe比計算の改善:** 現状`returns = targets.mean(dim=-1)`で各バッチの平均リターンからSharpeを計算していますが [66] 、より厳密には**日次ポートフォリオリターンを累積**して計算すべきです。Batch＝日なら問題ありませんが、もしランダムバッチなら現在のSharpe計算は正しくありません。したがって**検証データ全体**で日を追ってポートフォリオリターンを算出し、その標準偏差・平均からSharpeを計算します。FinancialMetricsにある`compute_information_coefficient`や`compute_rank_ic`は日次集計をしていますので、Sharpe版の計算も自作できます。例えば全検証データについて、各日で「予測に基づいたロングショート戦略の実現リターン」を計算し、最後に平均/標準偏差を求めます。この処理を新たな関数にしてLightningの`on_validation_epoch_end`で呼び出し、`val_sharpe`をログに送ることを推奨します。

- **RankIC計算の導入:** FinancialMetricsの`compute_rank_ic`を使用して、**各検証エポック終了時**にRankICを算出します。やり方は、全検証バッチの予測と実績をリストに貯め、`np.concatenate`した上で日付リストとともに`compute_rank_ic`に渡すだけです [74] [75] 。Baselineでは検証完了後にIC/RankICをログ出力しています [76] 。ATFT側でも同様に「Val RankIC: X.XX」と出せるようにします。実装上、LightningのLoggerを使っているなら`self.log('val_rankic', value)`とすればTensorBoardやW&Bに記録されます。

- **勝率・MSEのログ:** 勝率（hit_rate）は既に`val_hit_rate`として記録しています [6] 。計算方法は問題ないので、そのまま使います。MSEについてはQuantile LossやHuber Lossとは別に、**指標計算用にMSEを算出**します。例えばPyTorch LightningのMetricとして`MeanSquaredError`を定義するか、簡単には`loss_mse = F.mse_loss(predictions, targets)`をvalidation_stepで計算し`self.log('val_mse', loss_mse)`とします。こうすれば各エポックのMSEが記録されます。

- **非専門家向け指標の提示:** 上記の計算結果は、運用時にはレポートやダッシュボードで共有することになるでしょう。その際、**Sharpe比と勝率**は特に注目してもらうと良いです。Sharpeは年率換算値であることを伝え、勝率は「概ね○%の確率で方向を当てる」と説明します。RankICは必要に応じ「スコアと実績の相関」として補足します。MSEは直感的ではないため、「目標リターンとの平均誤差」とだけ述べ、**過去モデルとの比較**を中心に説明すると理解されやすいです。

総じて、**Sharpe・勝率・RankIC・MSEを併用した評価**を行うことで、モデルの強みと弱みを多面的に把握できます。例えば「勝率は若干50%台半ばだがRankICは0.1近くあり、予測強度に意味がある」「Sharpeは1を超えているのでリスクを取る価値がある」などの解釈が可能です。このような評価指標の運用により、非専門家含むステークホルダーに対してもモデル性能を**実務に結びつけてわかりやすく**説明・判断してもらえるでしょう。

参考文献・ソースコード:　　今回の提案にあたり、プロジェクト内の実装やドキュメントを参照しました。ATFT-GAT-FANモデルのアーキテクチャ概要は実装コメント [1] にあります。損失関数設計や環境変数による最適化策はプロジェクトドキュメント [58] [27] および実装コード [32] に詳細があります。BaselineのLightGBM実装 [24] と評価指標コード [15] [71] は、指標選定や正規化手法の有用な参考になりました。以上を踏まえ、提案内容を実装・調整いただくことで、本プロジェクトの多ホライズン株価リターン予測モデルの性能向上に繋がれば幸いです。

---

[1] [2] [3] [4] [6] [7] [8] [9] [18] [19] [20] [37] [41] [44] [52] [53] [59] [66] [67] [69] atft_gat_fan.py
https://github.com/wer-inc/gogooku3/blob/11e828b7d27488a6c6e5c4af04da0d37d7fd6438/src/atft_gat_fan/models/architectures/atft_gat_fan.py

[5] [10] [14] [16] [17] [36] [45] [46] [50] [55] [65] atft_gat_fan.yaml
https://github.com/wer-inc/gogooku3/blob/11e828b7d27488a6c6e5c4af04da0d37d7fd6438/configs/model/atft_gat_fan.yaml

[11] [12] [13] TODO.md
https://github.com/wer-inc/gogooku3/blob/11e828b7d27488a6c6e5c4af04da0d37d7fd6438/TODO.md

[15] [70] [71] [72] [74] [75] financial_metrics.py
https://github.com/wer-inc/gogooku3/blob/11e828b7d27488a6c6e5c4af04da0d37d7fd6438/src/metrics/financial_metrics.py

[21] [22] [29] [30] [32] multi_horizon_loss.py
https://github.com/wer-inc/gogooku3/blob/11e828b7d27488a6c6e5c4af04da0d37d7fd6438/src/losses/multi_horizon_loss.py

[23] [26] [27] [28] [34] [35] [47] [49] [54] [56] [57] [58] [60] [61] [62] [63] [64] ATFT_CRITICAL_ENV_VARS.md
https://github.com/wer-inc/gogooku3/blob/11e828b7d27488a6c6e5c4af04da0d37d7fd6438/docs/ml/atft/ATFT_CRITICAL_ENV_VARS.md

[24] [40] [42] [43] [68] [73] lightgbm_baseline.py
https://github.com/wer-inc/gogooku3/blob/11e828b7d27488a6c6e5c4af04da0d37d7fd6438/src/models/baseline/lightgbm_baseline.py

[25] [33] [48] train_atft_wrapper.py
https://github.com/wer-inc/gogooku3/blob/11e828b7d27488a6c6e5c4af04da0d37d7fd6438/scripts/_archive/train_atft_wrapper.py

[31] [38] [39] [76] train_atft.py
https://github.com/wer-inc/gogooku3/blob/11e828b7d27488a6c6e5c4af04da0d37d7fd6438/scripts/train_atft.py

[51] start_training.py
https://github.com/wer-inc/gogooku3/blob/11e828b7d27488a6c6e5c4af04da0d37d7fd6438/start_training.py