ExperATAI: Eine Browser-Anwendung für das experimentelle Anwenden von KI-Methoden im Algorithmic Trading

Lastenheft

Verfasser: Henning Beyer
Klasse: A20/2
Schuljahr: 2022/23
Datum: Leipzig, 10.03.2023

Inhaltsverzeichnis

1. Einleitung

Mit diesem Lastenheft werden die Anforderungen an das zu erstellende Programm zum Long-Time-Series-Forecasting (LTSF) von Crypto-Preisverläufen beschrieben.

Generell bietet sich für jede Person die Möglichkeit mit eigenen Kapital Geld in vielversprechende Aktien anzulegen und diese über einen längeren Zeitraum von einer Woche oder mehreren Jahren einzubehalten und Gewinne aus einem längerfristigen Preisanstieg zu beziehen. Jedoch erzielt man heutzutage über das modernere *Algorithmic Trading* höhere Gewinne in einem kürzeren Zeitraum und setzt sich bei einer niedrigeren Kaufs- und Verkaufsfrequenz einem viel kleineren Risiko von Preiseinbrüchen aus. Stattdessen kann man Handelsprozesse automatisieren und sich von der hauptsächlich manuell durchgeführten Fundamental-Analyse nun ganz auf die automatisierbare technische Analyse konzentrieren.

Mit den heutigen technischen Fortschritten ergibt sich neben dem *High-Frequency-Trading* die Möglichkeit zur Einbindung von KI-Modellen, um Handelsentscheidungen basierend auf technischen Indikatoren und relevanten Daten-*Features* zu treffen. Nur ist es fast immer der Fall, dass von Trading-Experten ausschließlich ältere weniger effektive, bewährte KI-Methoden verwendet werden und selbst in der Fachliteratur auf ein wichtiges *Feature-Engineering* fast immer fehlt. Demnach – und weil hierzu keine vollständige Lösung in der Literatur vorliegt – sollen in diesem Projekt die aktuell effektivsten KI-Methoden zum LTSF und ein allgemeingültiges Feature-Engineering implementiert und getestet werden, um daraus in einem weiteren Projekt basierend auf den Erfahrungen von diesem Erstprojekt am Ende ein funktionsfähiges Algorithmic-Trading-Programm zu erstellen. Die Anforderungen des Erstprojektes werden nun im Lastenheft beschrieben.

2. Beschreibung des Ist-Zustands

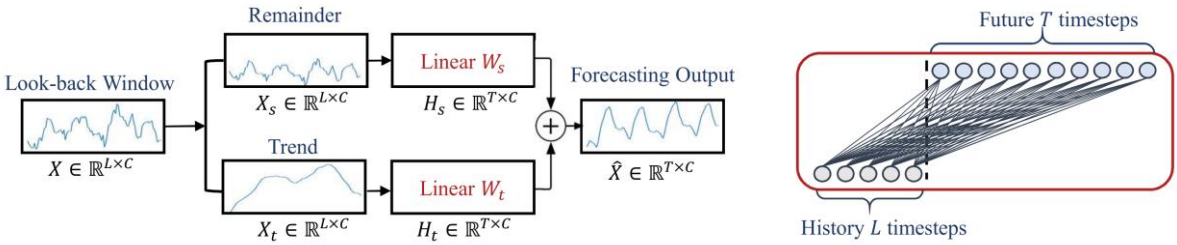
2.1 Dokumentanalysen zum Stand der KI-Methoden

Bezüglich dieser Anforderungen wurde zuerst selbst nach den passenden KI-Modellen für recherchiert, um dessen Auswahl entsprechend zu beschränken. Nämlich erweist sich ein Großteil der Modelle der Forschung als zu kompliziert und schwerfällig bei ebenfalls ungenügender Vorhersagegenauigkeit.

Im Abstand von nur einem Monat erschienen im Juni 2022 gleich zwei neue und zugleich simple Modelle für das LTSF, die mit einfacher neuronaler Struktur klar die Leistung von den bisher effektiven, jedoch rechenintensiven Transformer-Modellen übertreffen: DLinear [1] und LightTS [2]. Diese erweisen sich in den Forschungsberichten für den Einsatz bei Zeitreihen mit saisonalen Mustern sehr effektiv und können relativ genau über 512 zukünftige Datenpunkte in einem Berechnungs-Durchgang vorhersagen.

Generell ist das Prinzip von DLinear einfacher und auch das Modell mit nur zwei neuronalen Schichten zugleich sehr effizient und leicht zu implementieren. Eine gegebene Zeitreihe wird in diesem Modell in eine Trend- und eine Remainder-Komponente auftrennt, wobei jeweils eine neuronale Schicht die Charakteristiken jeweils einer Komponente erlernt und vorhersagt.

Schließlich wird aus beiden vorhersagen die originale Zeitreihe durch Addition wie in Abb. 1rekonstruiert:



(a) The whole structure of *DLinear* (b) One Linear Layer

Abb. 1: Die Abbildung des gesamten DLinear-Modells (a) mathematisch und (b) als neuronale Netz-Struktur [1].

Hiergegen betrachtet LightTS die Zeitliche Struktur genauer und entnimmt mit mehreren Teilnetzen über ein *Continuous-* und *Interval-Sampling* temporale Informationen, um diese mit einem *Information-Exchange-Block* zur Vorhersagebildung zu kombinieren. Diese aufwendigere Struktur ist womöglich nicht vollständig parallelisierbar und zu aufwendig zu implementieren, weswegen DLinear bevorzugt in dieser Anwendung zu implementieren ist. Vergleichbar gute und simple Modelle

2.2 Beobachtungen zum Feature-Engineering

Das Feature-Engineering ist für die Modell-Leistung mitunter der wichtigste Schritt, um dessen Leistung zu erhöhen. Persönlich wurde die Beobachtung gemacht, dass direkt von der Zeitreihe entnommene Features mehr Informationsgehalt als jegliche externen Features haben, die meist sehr aufwendig und mit sehr viel Zeitaufwand gewonnen werden müssen. Demnach ist es hier sinnvoll aus ausschließlich der gegebenen Zeitreihe technische Features zu extrahieren anstatt umständlich aus anderen Informationsquellen Informationen an die Data-Pipeline anzubinden. Somit wird die Fundamental Analyse für zu aufwendig gehalten und die technische Analyse klar bevorzugt, da sie genaue Informationen über Trend und Momentum der Zeitreihe liefert.

Hierbei ist zu beachten, dass das originale DLinear-noch kein Feature-Engineering betreibt, sondern jede Zeitreihe ausschließlich mit deren Daten vorhersagt. Dennoch soll ein Feature-Engineering im Sinne des Prototypings für andere LTSF-Modelle implementiert werden, vor allem da diese Pipeline des Feature-Engineerings auch für die RL-Modelle wiederverwendet werden kann.

3. Beschreibung des Soll-Konzepts

Mit dem zu erstellenden Programm soll primär experimentiert werden und eine Testumgebung für die Anwendung neuer KI-Methoden geboten werden, um sich selbst und andere Mitarbeiter mit den neusten KI-Methoden vertraut zu machen und die Konzepte neuer Methoden auf Anwendbarkeit zu testen. Dabei soll diese Anwendung noch keiner Trading-Anwendung gleichen und innerhalb des Algorithmic Tradings nur die Implementation eines KI-Modells wie DLinear bieten. Persönlich soll herausgefunden werden, ob DLinear für die Nutzung im Betrieb geeignet ist und andere Mitarbeiter sollen mithilfe dieser Anwendung erst überzeugt und gegebenenfalls danach geschult werden können.

Im Bereich des Algorithmic Trading gibt es mit der Zeit ständig neue Konzepte die selbstständig ausprobiert werden müssen, wie ebenfalls das Trading mit *Reinforcement Learning* (RL) Modellen, das zwar komplizierter als das Trading mit LTSF-Vorhersagen ist, dennoch aussichtsreicher als das Trading mit LTSF-Vorhersagen erscheint. Deshalb soll genau dasselbe Programm später in einem Zweitprojekt mit RL-Systemen unter derselben Schulungsfunktion erweitert werden. Optional kann die Erweiterung um diese RL-Systeme aber auch im Erstprojekt erfolgen.

Durch das benötigte Experimentieren ergeben sich weitere Kriterien der Anwendung, wie vor allem, dass mit der Anwendung erleichtert Experimente durchgeführt werden sollen, diese leicht reproduzierbar sind und dass Messwerte schnell aufgenommen und visualisiert werden können. Hierbei soll gleichzeitig durch die gute Reproduzierbarkeit und vor allem eine moderne, qualitative Oberfläche diesen Experimenten mehr Überzeugungskraft verliehen werden, wozu sich Python und die Bibliothek Plotly bestens eignen. Somit sollte die Anwendung anstatt in aufwendiger Kleinarbeit und bei veraltetem Design lieber übersichtlicher als Browser-Anwendung mit Plotly konzipiert werden.

Später existieren Pläne die effektiven Konzepte für das Algorithmic-Trading im Betrieb zu benutzen, weshalb die verwendeten Methoden möglichst performant und simpel implementiert werden sollten müssen. Womöglich wird diese modulare Anwendung, wie in der geplanten Anwendungs-Skizze in Abb. 2, um ein Echtzeit-Trading-System mit ähnlichem Aufbau- und Designs-Schema erweitert.

4. Beschreibung der Schnittstellen

Die Anwendung besitzt nur wenige Schnittstellen: Lediglich mit Maus und Tastatur sollen Eingaben für die auf dem Monitor angezeigte Benutzeroberfläche formuliert werden. An diese Oberfläche ist das Programm angebunden, das die Eingaben verarbeitet und wiederum die Ausgaben auf der Oberfläche visualisiert und auch im lokalen Verzeichnis hinterlegt. Teilweise wird diese Oberfläche in dieser Browser-Anwendung vom Browser gestützt, womit auch eine Schnittstelle von Programm zu Browser vorliegt.

Innerhalb des Programms gibt es weitere Software-Schnittstellen zwischen Daten, Datapipeline, KI-Modell und der Oberfläche. Eine Schnittstelle zum Internet kann existieren, wenn die Datensätze über die Oberfläche heruntergeladen werden, sollte aber ansonsten für die Experimente vermieden werden.

5. Funktionale Anforderungen

5.1 Bibliotheken und Datenspeicherung

Schon festgelegt wurde die Nutzung von Python und den Bibliotheken PyTorch zur Implementierung des DLinear-Modells sowie Plotly zur Umsetzung der Oberfläche. Zusätzlich wird festgelegt, dass die Datensätze möglichst dateiorientiert gespeichert werden, um diese einfacher und schneller auslesen zu können. Problematisch können hier nur große Datensätze von mehreren GB werden, die als CSV-Dateien sehr geringe Lese- und Schreibgeschwindigkeiten haben. Deswegen wird festgelegt alle Datensätze als PICKLE-Dateien zu speichern, damit die Lese- und Schreibdauer in diesem Fall von mehreren Minuten auf ein paar Sekunden reduziert wird und

der benötigte Speicherplatz potenziell halbiert wird. Allgemein wird geraten die Datensätze mit der Pandas-Bibliothek in ein übersichtliches Dataframe-Format zu laden und damit ebenfalls vereinfacht das Feature-Engineering vorzunehmen. Möglichst sollen auch Parameter-Dateien von Experimenten und die zur Wiederverwendung abgespeicherten Experiment-Ergebnis-Dateien gut auseinanderhaltbar sein. In der Anwendung soll vermieden werden die Modell-Gewichte zu speichern und die Experiment-Ergebnisse über die entsprechend abgespeicherten Dateien rekonstruiert werden.

5.2 Datensätze

Nach den ersten Überlegungen wird es 2 Arten von Datensätzen geben, mit denen zwei unterschiedliche LTSF-Experimente durchgeführt werden. Einmal soll es einen *Scalping-Trading*-Datensatz geben, der einen längeren historischen Zeitraum mehrerer Crypto-Währungen in Minuten-Frequenz umfasst, von welchen aus Datensätze in niedrigeren Frequenzen mittels Downsampling umgewandelt werden können. Grund für die Wahl der Crypto-Datensätze für das LTSF ist dessen gute Verfügbarkeit, dessen umfassender Datensatz-Größe sowie natürlich auch die Relevanz der der Crypto-Daten an sich für das weitere Programm.

Da es mitunter wirklich sehr umständlich ist die richtigen Datenquellen für historische Crypto-Datensätze zu finden, mit welchen Datenbestände über mehrere Jahre in Minuten-Frequenz ohne Beschränkungen abgerufen werden können, wird deshalb hierzu ein Python-Skript mitgegeben, mit dem die anforderungsgerechten historischen Datensätze in monatlichen Zeitfenstern angefragt werden können. Hierbei handelt es sich um Datensätze mit der klassischen *Open-High-Low-Close-Volume*-Struktur (OHLCV).

Beim zweiten *High-Frequency-Trading*-Datensatz (HFT) soll es sich um einen in Echtzeit abgefragten und zusammengestellten Datensatz im Millisekunden- bis Sekundenbereich handeln, für den zusätzlich neben den OHCLV-Angaben auch die wesentlichen Angaben über die Bid- und Ask-Preise mit angegeben werden. Grundsätzlich unterscheidet sich das Scalping-Trading und das HFT sehr voneinander, weshalb es wie in Abb. 2 zu zwei separaten Experimenten mit dem jeweiligen Datensatz kommen soll.

Vorgegeben wird für die Erstellung des Datensatzes lediglich, dass die Daten über eine Binance-Python-Schnittstelle angefragt werden müssen und dass die auftretenden Latenzen im Millisekunden-Bereich berücksichtigt werden. Nach einem Versuch mit der Binance-API besteht mit HTTP-Requests eine Request-Dauer von etwa 240ms die für das Erstprojekt ausreichend ist. Optional kann diese Latenz-Zeit auf 10ms pro Request reduziert werden, indem die Anfragen von einem Server nahe des Binance-Serversitzes in Tokyo gesendet und empfangen werden. Ansonsten ist geplant diese Funktionalität mit einer akzeptableren Frequenz im folgenden Zweitprojekt mithilfe von Amazon Web Services umzusetzen und die Skriptausführung zur Datensatz-Erstellung auszulagern.

5.3 Feature- und Indicator-Engineering

Das Feature-Engineering ist ein oft vernachlässigter Prozess der Datenaufbereitung für KI-Modelle, mit dem aus den OHLCV-Features weitere zusätzliche Features als relevante Informationen für das KI-Modell gewonnen werden. Beispielsweise lässt sich für Zeitreihen ein Moving Average (MA) oder ein Log-Return berechnen und aus dem Log-Return wiederum ein MA-Log-Return, wobei diese Features dem Modell eher sagen, wie sich ein zukünftiger Preisverlauf

entwickeln könnte. Insgesamt ist dieser Prozess des Feature-Engineerings immer ein langwieriger Prozess, der stets durch viel experimentelles Probieren verfeinert werden muss. Deshalb werden hierzu als Orientierung vorab bewährte einige Techniken des Feature-Engineerings geteilt. Insgesamt sollen für die Erstellung der Datensätze etwa 100 verschiedene Features durch den Nutzer berechnet werden können und die Features möglichst kompakt und nachvollziehbar in Feature-Gruppen unterteilt werden. Eigene effektive Konzepte dürfen immer mit eingebracht werden.

Dazu werden im Voraus diese folgenden Methoden des Feature-Engineerings als Arbeitsgrundlage vorgegeben:

- Grundlegende Feature-Interactions. Zwei Features werden mit den Operatoren - und * kombiniert. Seltener sind die Operatoren + und / zu verwenden.
- Log-Features. Diese dienen eher als Normalisierung und Stationarisierung der Zeitreihe. Die Logarithmus-Funktion wird auf ein Feature angewendet.
- MA- und Exponential-MA-Features (EMA). Sie geben Auskunft über den Trend-Verlauf.
- Lag-Features. Hiermit ist gemeint zeitlich zurückliegende Feature-Werte als Input zu benutzen. Wenn DLinear implementiert wird, werden diese Lag-Features nicht benötigt.
- Lag-Differenzen bzw. Lag-Returns. Diese sind praktisch, um das Momentum der Zeitreihe zu erfassen. Z.B. kann zum aktuellen Close-Preis die Differenz zu mehreren zurückliegenden Close-Preisen gebildet werden.
- Feature-Engineering höheren Grades. Features, welche aus einem Feature-Engineering hervorgehen werden nochmals einer Methode des Feature-Engineerings unterzogen, um den Daten spezifischere Informationen zu entnehmen. Beispiele hierfür sind MA-Differenzen und EMA-Differenzen für unterschiedlich große MA-Zeitfenster sowie MA-Returns, Log-Returns und MA-Log-Returns.

Für beide Datensätze ist ein leicht unterschiedliches Feature-Engineering zu codieren, da für den Live-Datensatz die *Bid-Ask-Features* dringend in das Feature-Engineering mit einbezogen werden müssen und durch die hohe Frequenz andere MA-Zeitfenster zu wählen sind. Dennoch sollten die oben genannten Features grundsätzlich für beide Datensätze bei verschiedenen Zeitfenstern genutzt werden können. Ein *Feature-Scaling* darf natürlich nie nach dem Feature-Engineering vergessen werden.

Das was vom Auftraggeber als *Indicator-Engineering* bezeichnet wird, umfasst lediglich das Berechnen der typischen technischen Indikatoren, wie z.B. den VWAP, den ADX oder den RSI aus den OHLCV-Features. Da aber zu erwarten ist, dass die Indikatoren bis auf Ausnahmen eher wenig relevante Informationen zum LTSF liefern, ist für dieses LTSF gar kein Indicator-Engineering zu implementieren. Nur einzelne informative Indikatoren können trotzdem zusätzlich implementiert werden, wobei die MA-, EMA- sowie die MACD-Features schon solche technischen Handelsindikatoren darstellen.

Eher dienen diese Indikatoren zum Treffen von Handelsentscheidungen und ein vollständiges Indicator-Engineering ist ausschließlich für das optionale RL-System zu entwickeln, das den historischen Datensatz mit niedrigerer Frequenz verwendet. Hier möchte der Auftraggeber erwähnen, dass für den Datensatz im Millisekunden-Bereich fast keine Indikatoren verlässlich funktionieren und für diesen Datensatz auf das Indicator-Engineering überwiegend verzichtet werden kann.

5.4 Anforderungen an die Implementation des KI-Modells

Allgemein sollen die Modelle nach der Original-Implementationen des Forschungsberichtes für speziell für die Anwendung erneut implementiert werden. Begründete Anpassungen am Modell sind möglich wie ebenfalls die begründete Wahl eines alternativen Modells sind, wenn dieses die Anforderungen mindestens im selben Maße erfüllt.

Für das optionale RL-System ist das Modell der *Proximal Policy Optimization* (PPO) [3] zu wählen, das nach einer eigenen Dokument-Analyse das effektivste RL-Modell-Konzept neben der *Podracer*-Architektur [4] zu sein scheint, die wegen der Anforderungen an die Rechenleistung nicht zu implementieren ist. Für PPO ist eine gut begründete Reward-Funktion zu wählen und ein geeignetes RL-Environment zu verwenden, wobei sich eine vollständige Implementation des PPO-Modells innerhalb der Bibliothek FinRL [5] befindet. Es wird empfohlen diese Bibliothek zu verwenden.

Wichtig für die KI-Modell-Implementationen ist ein parallelisierter Modell-Code mittels der Verwendung von PyTorch-Tensoren, damit im Training das Batching und die Nutzung von CUDA-GPUs unterstützt wird. Ein Multiprocessing ist ebenfalls erwünscht und sollte aber von den Bibliotheken übernommen werden. Es sind unbedingt beide Modelle in der Projektbegleitenden Dokumentation zu dokumentieren, egal ob das RL-System implementiert wird oder nicht, denn es soll wenigstens eine Planungsgrundlage für die Implementation des RL-Systems für das Zweitprojekt vorab geben.

Als gute, vollständige Übersicht zum Stand der RL-Methoden und RL-Anwendungsbereiche im Algorithmic Trading wird die folgende Quelle sehr empfohlen, da diese auch einige Grundbegriffe und -Konzepte erläutert: [6]. Außerdem wird mit diesen Forschungsberichten [7], [8] auf einige gute Lösungsansätze für RL-Systeme mit dem PPO-Modell hingewiesen. In der Quelle [8] wurde sogar ebenfalls eine PPO-Implementation mit einem LSTM-Modell vorgenommen, die bessere Leistungen erzielt, da LSTMs die zeitliche Struktur der Zeitreihen besser erfassen. Nach diesem Vorbild soll in der Dokumentation ein Vorschlag für die Modell-Struktur von PPO erfolgen. Vielleicht lässt sich die Struktur von DLinear hier einbauen, sodass das Konzept von PPO mit mehreren KI-Strukturen getestet werden kann. Wegen dieser Menge an der noch in der Forschung nicht ausgearbeiteten Theorie wurde die Entwicklung des RL-Systems als optional erklärt, aber dessen Planung wird weiterhin gefordert.

5.5 Anforderungen an die Oberfläche

Die Oberfläche der geforderten Browser-Anwendung soll mit der Plotly-Bibliothek umgesetzt werden und dabei möglichst übersichtlich und intuitiv gestaltet werden. Hierbei bietet Plotly alle benötigten Komponenten und der Browser bietet die Möglichkeit vereinfacht informative Dashboards zu erstellen. Wichtig hinsichtlich der Nutzerfreundlichkeit ist es ebenfalls, mit so wenigen Elementen und in einer so einheitlichen und kompakten Form wie möglich, alle erforderlichen Informationen und Funktionen des Programmes auf der Oberfläche darzustellen. Letztlich soll die Oberfläche nach der in Abb. 2 dargestellten Programm-Struktur umgesetzt werden.

Der Grund für die hohen Anforderungen an die Oberfläche mit einem erwarteten Hochglanz-Design liegt an der Tatsache, dass hinderliche Systeme die Effektivität und den Erfahrungswert des Programms massiv einschränken und dessen Nutzen selbst bei einer makellosen Implementierung der KI-Methoden gänzlich zerstören. Indem das Projekt übersichtlich und

einfach bleibt, können dessen Bestandteile besser verstanden, angewendet und für den nächsten Prototyp abgeändert werden sowie viel Zeit gespart werden.

6. Programm-Funktionalitäten

6.1 Data-Pipeline

Generell soll der Nutzer nach der Abb. 2 von einer Start-Seite auf die gewünschten Programm-Funktionalitäten wie hier die Experimente mit den LTSF- und RL-Systemen. Während der schrittweisen Festlegung aller Experimentbedingungen, beginnt der Nutzer ggf. zuerst mit dem Herunterladen bzw. mit dem Aktualisieren der hinterlegten Datensätze. Anschließend legt dieser über ein weiteres Fenster diese wichtigen Experiment-Einstellungen für den Datensatz vor:

- **Konfigurationsmöglichkeiten des verwendeten Datensatzes:**
  - Die Datensatz-Frequenz:
    - Von der gegebenen Ein-Minuten-Frequenz kann der Datensatz in eine beliebige Frequenz von z.B. 5 Minuten oder einer Stunde umgewandelt werden.
    - Möglich soll die Auswahl der Frequenzen (1 *min*, 2 *min*, 5 *min*, 10 *min*, 15 *min*, 30 *min*) sein, aber durchaus auch die niedrigen Frequenzen von (1 *h*, 2 *h*, 6 *h* 12 *h* und 1 *d*) festgelegt werden.
  - Das Datensatz-Start- und End-Datum
    - Dabei wird die absoluten Trainingsdatensatz-Größe angezeigt.
  - Der Relative Trainings- und Test-Datenanteil
    - der Validierungs-Datenanteil entspricht immer genau dem Test-Datenanteil, um eine konstante Validierungs-Voraussetzung zu gewährleisten.
    - Der Testanteil liegt immer zeitlich hinter dem Trainings-Daten-Anteil.
    - Es findet kein *Shuffling* der Trainingsdaten statt und für jede Epoche werden die Trainingsdaten von Anfang bis zum Ende konsekutiv erlernt.
  - Das Feature-Engineering:
    - Hier sollen keinesfalls alle bis zu über 100 Features einzeln per Hand festgelegt werden. Stattdessen werden über Feature-Gruppen-Namen jeweils etwa 1 bis 10 Features mit einer Auswahl ausgewählt.
    - Der Nutzer soll hier nicht zu filigrane Einstellungen vornehmen können und damit auch keine einzelnen Features der Gruppen abwählen können.
  - Die Feature-Scaling-Methode:
    - Auswahl zwischen dem *Standard-Scaling* und keiner Daten-Skalierung

6.2 Definition und Durchführung des Modell-Trainings

Nachdem die gesamte Data-Pipeline konfiguriert ist, werden schließlich Modell- und Benchmark-Parameter in einer Oberfläche festgelegt. Hierbei sind folgende Einstellungen zu definieren:

- **Allgemeine Parameter für das Modell-Training:**
  - Die Auswahl der GPU; die Auswahl keine GPU zu nutzen
  - Die Trainings-Batch-Size

    - Die Trainings-Epochen
    - Die Learning-Rate
    - Die Patience für das Early-Stopping

- **Die Fehlermetrik für das Training des LTSF-Systems:**
    - Zur Auswahl steht nur der mittlere absolute Fehler der Mean-Squared-Error (MSE), mit welchem besonders große Vorhersagefehler bestraft werden.
    - Es sind beabsichtigt keine anderen Trainings-Metriken zu verwenden, um die Auswahlmöglichkeiten klein genug zu halten.

- **Die Modell-Parameter für DLinear**:
    - Das Look-Back-Window
    - Das Vorhersagezeitfenster
    - Das MA-Decomposition-Zeitfenster; beträgt standardmäßig 25
    - Es ist immer eine univariate Vorhersage des Close-Preises zu treffen und zur Einfachheit keine multivariate Vorhersage einiger der vorhandenen Features. Damit ist der Parameter *individual Layers* immer auf *false* gestzt, womit nur das Modell *DLinear-S* benutzt wird und nicht das Modell *DLinear-I*.

Danach wird über eine Schaltfläche das Modell-Training mit epochenweiser Validierung eingeleitet, wobei im Modell-Training folgende wichtige Zwischen-Informationen angezeigt werden sollen:

- **Übergangsinformationen für Während des Modell-Trainings:**
    - Voraussichtliche Dauer in Sekunden; Fortschritt in Epochen
    - Hardware-Auslastung

Nach dem längeren oder kürzeren Training folgt eine Test-Phase mit üblicherweise 30 % der Gesamtdaten für die folgendes vorübergehend angezeigt werden soll:

- **Übergangsinformationen für Während der Modell-Testung:**
    - Voraussichtliche Dauer in Sekunden; Fortschritt in Prozent

6.3 Auswertung der Modell-Benchmark

Schließlich gelangt man nach dem Testen zu dem Test- bzw. Benchmark-Ergebnissen, die als Gesamtheit in einem informativen und interaktiven Dashboard angezeigt werden sollen. Hierbei sollen auf diesem Dashboard angezeigt werden:

- **Modell-Laufzeit- und Ressourcenverbrauch-Statistiken:**
    - Maximaler RAM-verbrauch; maximale CPU und GPU-Auslastung
    - RAM-Verbrauch alleinig durch das Modell
    - Laufzeit der Modell-Ausführung in Millisekunden mit 2 Nachkommastellen
    - Anzahl der Modell-Parameter

- **Angaben der Experimentbedingungen:**
    - Entsprechend der Parameter-Vergabe ist der Modell-Name aus der Literatur hervorgehoben anzuzeigen. Hier wären das DLinear und PPO. Bei Abwandlungen des Modell-Konzepts durch die Parameter-Vergabe ist das dringend bei z.B. dem LSTM-PPO zu benennen.
    - CPU: Name und nur bei Nutzung die Anzahl der Kerne und Threads

    - GPU: Name und GPU-Memory
    - Arbeitsspeicher: RAM in GB
    - Wichtige Trainings-Parameter: Batch-Size, Anzahl der Epochen und Learning-Rate Wichtige Parameter von DLinear bzw. PPO
    - Angaben zum Datensatz:
        - Datensatz-Frequenz
        - Start- und End-Datum
        - Absolute Anzahl der Datenpunkte
        - Relativer Training- und Test-Anteil
        - Anzahl der Features
        - Feature-Scaling-Methode

- **Trainings- und Validierungs-Loss gemeinsam in einer Learning-Curve:**
    - Für jede Epoche soll ein Datenpunkt für Trainings- und Test-Fehler visualisiert werden.
    - Somit werden insgesamt Trainings- und Test-Fehler in einer eigenen Verlaufskurve miteinander dargestellt.
    - Dieses Prozedere erfolgt für sowohl für das LTSF- als auch das RL-System.

- **Visualisierung der Modell-Leistung des LTSF-Systems:**
    - Hier soll ein interaktives Linien-Diagramm mit den Diagramm-Achsen für den mittleren absoluten Fehler (MAE) und die Vorhersage-Weite dargestellt werden.
    - Es soll unbedingt der MAE für jede Vorhersage an der Stelle *n* unabhängig von den Vorhersagen anderer Stellen berechnet werden.

- **Visualisierung der Modell-Vorhersagen des LTSF-Systems:**
    - Ein Interaktiver Graph mit welchem der Nutzer an beliebigen Zeiträumen die Close-Preise mit den Modell-Vorhersagen visuell vergleichen kann.
        - DLinear berechnet pro Bezugspunkt *n* Vorhersagen. Nutzer können jeweils jede erste bis *n*-te Vorhersage im Vergleich zu den Close-Preis angezeigt werden. Hier sollte die Auswahl auf Fünfer-Schritte eingeschränkt werden, sodass z.B. jede
        - Im Falle des LTSF können zusätzlich vom Nutzer einzelne Zeitpunkte ausgewählt werden, von denen aus alle *n* dazugehörigen Modell-Vorhersagen angezeigt werden.

- **(Optional) Eine Nachbetrachtung der Features in einem weiteren Fenster:**
    - Zu den Benchmark-Ergebnissen kann zurückgekehrt werden
    - Visualisierung der *Feature-Importances* in einem interaktiven Kreisdiagramm
        - Möglichkeit zur Auswahl von Pearson- und Spearman-Correlation
        - Zu irrelevante Features werden zu einer Gruppe zusammengefasst.
    - Selektieren und Visualisieren einzelner Features
        - Visualisieren in einem Linien-Diagramm
        - Visualisieren in einem Candle-Stick-Diagramm
        - Visualisieren in einem Streu-Diagramm
    - (Optional) Dieses Fenster kann sowohl nach der Benchmark als auch während der Datensatz-Konfiguration nach dem Feature-Engineering eingesehen werden.

Um diese Bestandteile der Benchmark zu visualisieren ist es damit auch erforderlich neben den instanziierten Datensatz ebenfalls die Vorhersagen als Dataframe-Objekt zu instanziieren. Möglichst sollten dabei nicht mehr als 8 GB RAM verbraucht werden, sodass die Anwendung auf

gewöhnlichen Geräten läuft. Durch die sehr einfachen und kleinen Modelle sollte dies möglich sein.

Andererseits sollen die Messergebnisse und Experiment-Parameter in einem gut unterscheidbaren Format abgespeichert werden, sodass nach den Experimenten alle Messergebnisse im Notfall eingesehen werden können oder sogar mit der oben benannten optionalen Funktionalität wieder auf dem Dashboard visualisiert werden können. Dazu soll zur Übersichtlichkeit für den Nutzer keine Speicherung der Modell-Gewichte erfolgen, da die Experimente dann sowieso visualisierbar und durch das einfache Modell schnell reproduzierbar sein sollten. Jegliche Datensätze und Vorhersagen sind als PICKLE-Dateien zu speichern und alle Ergebnisse, Metriken und Experiment-Konfigurationen in den sehr übersichtlichen YAML-Dateien. Optional wird vorgeschlagen von jedem Dashboard einen standartmäßigen Screenshot als PNG-Datei abzuspeichern, wenn dies umstandslos geht.

7. Nicht-Funktionale Anforderungen

Mitunter teilweise benannt und erläutert werden die nicht-funktionalen Anforderungen hier zusammengefasst aufgelistet:

Absolut zu erfüllende Anforderungen:

- Zuverlässigkeit des Systems
- Hohe Nutzerfreundlichkeit
- Komplett korrekte Experiment-Messwerte
- Modularität des Systems wie nach Abb. 2

Im bestmöglichen Maße zu erfüllende Anforderungen:

- Abstraktion der Programm-Bestandteile für die Erweiterbarkeit
- Dadurch sollen Programm-Bestandteile eine hohe Wiederverwendbarkeit für die Weiterentwicklung erhalten
- Korrigierbarkeit und Wartungsfähigkeit mittels eines standardisierten und übersichtlichen, kommentierten Codes

Es wird davon ausgegangen, dass sehr grundlegende Prinzipien erfüllt werden, indem der Code übersichtlich mithilfe des OOP codiert wird und Code-Dateien mit jeweils zusammengehörenden Klassen sinnvoll aufgetrennt in einer logischen Struktur im Verzeichnis hinterlegt sind, was als Lokalität gezählt wird.

8. Risikoakzeptanz

Generell muss unbedingt das, was im Lastenheft als Anforderung definiert wurde, vollständig erfüllt werden, damit die spätere Planung definitiv ab den grundlegenden Punkt der Fertigstellung der Programm-Struktur und der Erfahrungen mit dem LSTF-Modell weitergeführt werden kann. Wegen des hohen Risikos die neuen KI-Methoden in das Projekt einzubinden wurden die Ziele des Erstprojektes für eine lieber klar durchführbare Planung eingeschränkt. Andererseits wurde wegen des hohen Erfahrungsbedarfs für spätere Entscheidungen eine schrittweise Entwicklung des Projektes als experimentelles Prototyping bevorzugt und ein längerer Zeitrahmen zur sorgsamen Erstellung des Gesamtprojektes eingeplant. Diese Entscheidungen liegen in der folgenden Risikoanalyse begründet.

8.1 Risikoanalyse

Es steht zwar viel Zeit zur Entwicklung der Anwendung zur Verfügung, jedoch besteht durch die hohe Schwierigkeit der Planung ein hohes Risiko sich zu verplanen und erneut eine langwierige Änderung in der Programm-Struktur vorzunehmen. Deshalb wurden grundlegende Methoden zur Datenbeschaffung und die Wahl der KI-Methoden im Vorfeld vorgenommen und ein Puffer an das Ende des Arbeitszeitraums gelegt. Dabei ist zur Risikominimierung erforderlich, dass möglichst alle verpflichtenden Anforderungen vor dem Puffer erledigt sein müssen und dass schon Teile der optionalen Anforderungen zur Erleichterung des schwierigeren Zweitprojektes erledigt werden.

Hinsichtlich der Ressourcen besteht dagegen kein unmittelbares Risiko, da zur Entwicklung keine finanziellen Mittel oder kostenpflichtige Hilfssoftware benötigt wird und Lieferzeiten wegfallen.

Dagegen besteht eine hohe Schwierigkeit der Entwicklung des RL-Systems, durch welches ein hohes Verplanungsrisiko besteht, da nicht nur immer noch das RL-System geplant werden muss und die Prinzipien des RL-Bereiches dem Auftraggeber noch z.T. unbekannt sind, sondern weil auch die Implementation im Gegensatz zum simplen DLinear-Modell höchst spezifisch mit eigener optimaler Reward-Funktion und einem eigenen RL-Environment stattfinden muss. Deshalb findet die Entwicklung des RL-Systems voraussichtlich im Zweitprojekt statt, für die die Planung und die vollständige theoretische Begründung des RL-Systems zur Risikominimierung im Erstprojekt erfolgen muss.

9. Skizze des Entwicklungszyklus und Programms

Die Entwicklung des aller Projekte als Gesamtprojekt orientiert sich am experimentellen Prototyping. Erst werden mit der Entwicklung von simpleren, dennoch auch brauchbaren Konzepten die grundlegenden Erfahrungen gesammelt, um hiermit die komplizierteren End-Ideen umzusetzen. Insgesamt ist am Ende des Gesamtprojektes geplant ein hochwertiges sowie einsatzfähiges RL-Trading-System mit funktionierender Binance-Anbindung zu entwickeln. Hierfür müssen jedoch zuvor die neuen KI-Konzepte ausprobiert und auf Anwendbarkeit untersucht werden. Diese darin einzelnen enthaltenen Entwicklungsstränge zur Entwicklung des LTSF-Systems, des RL-Systems sowie die abgewandelte Echtzeitanwendung dieser Modelle wie in Abb. 2, orientieren sich dabei jeweils an dem klassischen Lebenszyklus-Modell in Abb. 3, für das die Planung und die Spezifikation zum Teil erfolgt sind und die Programmierung und Testung auf Grundlage des Lastenheftes zu erfüllen sind. Somit folgt die Entwicklung des Erstprojektes auch dem klassischen Lebenszyklus-Modell in Abb. 3.

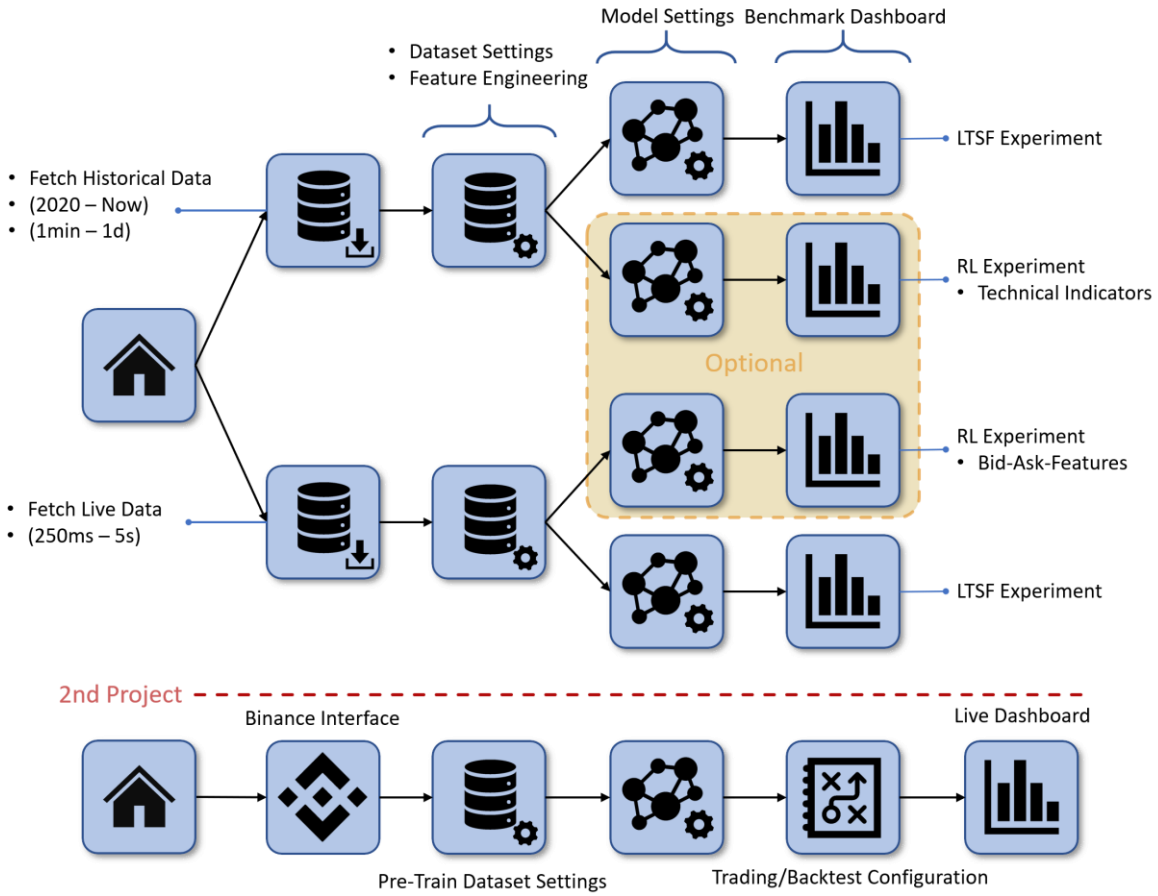


Abb. 2: Die vollständige Darstellung der geplanten Programm-Struktur für sowohl Erst- als auch Zweitprojekt. Diese ist unbedingt einzuhalten.

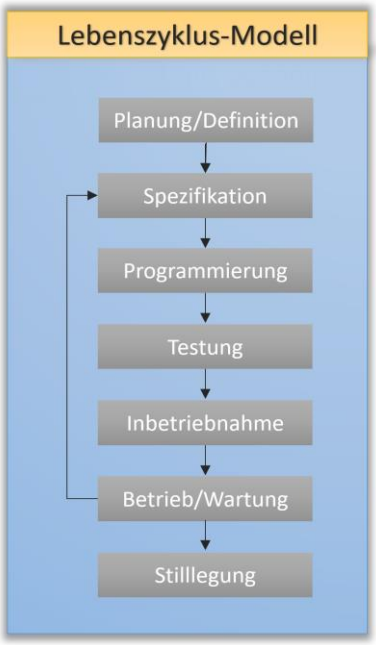


Abb. 3: Die Darstellung des Lebenszyklus-Modells, das für die Entwicklung einzelner Programm-Komponenten genutzt werden kann.

10. Abnahmebedingungen

Genau am 13.03.2023 um 11:45 ist das Projekt im Zimmer 118 der Arwed-Rossbach-Schule Leipzig mit einem USB-Stick zu übergeben oder alternativ in der LernSax-Schulcloud der Arwed-Rossbach-Schule im Gruppenverzeichnis *„A 20 IS1“* mit allen Dateien hochzuladen.

Eine Woche vor der Projekt-Abgabe soll das Programm dem Auftraggeber vorgestellt werden und der Fortschritt besprochen werden. Nach dem Abgabe-Termin soll das Programm der Schulklasse kurz präsentiert werden.

10.1 Lieferumfang

Zu liefern sind jegliche im Programm verwendete Code-Dateien, Installations-Skripts und Datensätze sowie jegliche mit der Dokumentation im Zusammenhang stehenden Messergebnisse, Konfigurationsdateien einschließlich der projektbegleitenden Dokumentation.

10.2 Abnahmekriterien

Kriterium	Erfüllung
Die Implementation von DLinear oder dessen Alternativ-Modell ist exakt und performant umgesetzt.	
Für die RL-Systeme wurden die Experimente geplant und die neuronale Struktur des PPO-Modells bzw. die neuronale Struktur von dessen Alternativ-Modell festgelegt.	
Eine vollständige projektbegleitende Dokumentation wurde erstellt.	
In der Dokumentation sind die Modelle PPO und DLinear oder deren Alternativen vollständig dokumentiert.	
Ein umfangreiches Feature-Engineering liegt vor mit etwa 100 Extra-Features.	
Es können mit dem Programm verschiedene LTSF-Experimente mit jeweils dem beschriebenen Scalping-Trading- und HFT-Datensatz durchgeführt werden.	
Bei der Anwendung handelt es sich um eine Browser-Anwendung, die mit den Installationsskripts auf jeden Windows-10-Rechner funktioniert.	
Das Programm hat eine gut erweiterbare, modulare Struktur und ist besonders nutzerfreundlich.	
Das Programm bzw. der entwickelte Programm-Teil repräsentiert die Programm-Struktur in Abb. 2.	
Die Oberfläche ist im hohen Maße qualitativ und übersichtlich aufgebaut.	

Alle verpflichtenden im 6. Abschnitt beschriebenen funktionalen Anforderungen wurden vollständig erfüllt. Alle unter dem 7. Abschnitt gelisteten nicht-funktionalen Anforderungen wurden bestmöglich eingehalten und die wichtigsten dieser Anforderungen erfüllt. Jede der zu liefernden Leistungen wird geliefert.	

Literaturverzeichnis

[1] A. Zeng, M. Chen, L. Zhang und Q. Xu, „Are Transformers Effective for Time Series Forecasting?“, arXiv:2205.13504v2, 2022.

[2] T. Zhang, Y. Zhang, W. Cao, J. Bian, X. Yi, Z. Shun und J. Li, „Less Is More: Fast Multivariate Time Series Forecasting with Light Sampling-oriented MLP Structures“, arXiv:2207.01186, 2022.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford und O. Klimov, „Proximal Policy Optimization Algorithms“, arXiv:1707.06347, 2017.

[4] M. Hessel, M. Kroiss, A. Clark, I. Kemaev, J. Quan, T. Keck, F. Viola und H. van Hasselt, „Podracer architectures for scalable Reinforcement Learning“, arXiv:2104.06272, 2021.

[5] X.-L. Liu, H. Yang, Q. Chen, R. Zhang, L. Yang, B. Xiao, C. D. Wang und b. c. a, „FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance“, arXiv:2011.09607, 2020.

[6] S. Sun, R. Wang und B. An, „Reinforcement Learning for Quantitative Trading“, arXiv:2109.13851, 2021.

[7] H. Yang, X.-Y. Liu, S. Zhong und A. Walid, „Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy“, SSRN: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3690996, 2020.

[8] S. Lin und P. A. Beling, „An End-to-End Optimal Trade Execution Framework based on Proximal Policy Optimization“, IJCAI: https://www.ijcai.org/Proceedings/2020/627, 2020.