

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

*Pflichtenheft*

Verfasser: Henning Beyer  
Klasse: A20/2  
Schuljahr: 2022/23  
Datum: Leipzig, 12.03.2023

# Inhaltsverzeichnis

<b>2. Dokumentation der KI-Methoden für das LTSF .....</b>	<b>3</b>
2.1 Linear und NLinear .....	3
2.2 DLinear und NDLinear .....	4
<b>3. Theorie und Entwurf des RL-Systems .....</b>	<b>5</b>
3.1 Grundlagen des Reinforcement Learnings .....	5
3.2 Beschreibung der verwendeten Binance-Umgebung .....	6
3.2.1 State, Action und Reward der Binance Umgebung.....	6
3.3 Das verwendete PPO-Modell .....	7
3.3.1 Mathematische Beschreibung.....	7
3.4 Entwurf des RL-Systems .....	8
3.4.1 Agent .....	8
3.4.2 Environment.....	8
3.4.3 Netzwerk-Strukturen des Agents .....	9
3.4.4 Bereits codierte Bestandteile des RL-Systems.....	9
3.4.5 DLinear-Varianten im RL-System .....	10
<b>4. Anwendungsbeschreibung.....</b>	<b>10</b>
<b>5. Dokumentation des Backends .....</b>	<b>12</b>
5.1 Backend-Struktur .....	12
5.2 Datensätze .....	12
5.3 Modell-Training .....	12
5.4 Experiment-Management-Klassen .....	13
<b>6. Dokumentation des Frontends .....</b>	<b>13</b>
6.1 Grundlegende Funktionen .....	13
6.3.1 Homepage.....	14
6.3.2 Datensatz-Konfiguration.....	15
6.3.3 Datensatz-Fetching.....	15
<b>7. Programm-Testungen .....</b>	<b>15</b>
<b>8. Erfüllung des Anforderungskatalogs.....</b>	<b>16</b>
<b>9. Fazit.....</b>	<b>17</b>
9.1 Weitere Planung.....	17
<b>Literaturverzeichnis .....</b>	<b>18</b>

# 1. Einleitung

In diesem Pflichtenheft wird das zu erstellende Programm, dessen Entwicklung sowie der Entwurf des RL-Systems beschrieben und dokumentiert. Daneben befindet sich eine selbstangefertigte Dokumentation der verwendeten KI-Modelle in diesem Pflichtenheft.

Basierend auf dem Lastenheft des Auftraggebers muss ein Programm mit einer sehr nutzerfreundlichen Oberfläche zum Durchführen von neuen Experimenten im Bereich des Algorithmic Tradings (AT) erstellt und mit diesem Pflichtenheft dokumentiert werden. Hierbei ist wegen dem Umfang und der Komplexität des finalen Programms das Gesamtprojekt des Auftraggebers in zwei Projekte unterteilt worden, von welchen mit diesem Pflichtenheft das erste dokumentiert und erfüllt wird.

In der Erfüllung der Anforderungen kommt es den Auftraggeber vor allem auf eine hochqualitative Oberfläche, ein simples Design und einer simplen, erweiterbaren, übersichtlichen Programmstruktur sowie einer ausgesprochen hohen Nutzerfreundlichkeit an, sodass Experimente leicht und flexibel mit dem Programm *ExperATAI* durchgeführt werden können. Die funktionellen Anforderungen an das Erstprojekt beziehen sich auf die Entwicklung eines *Long-Time-Series-Forecasting-Systems* (LTSF) mit dem erst neulich in der Forschung erschienenen Modell DLinear. Für das Zweitprojekt soll jedoch schon im Vorfeld im Erstprojekt ein Reinforcement-Learning-System (RL) entworfen werden.

Insgesamt wurden die Anforderungen des Auftraggebers erfüllt und qualitative Zusätze im Backend implementiert. Nur die letzte Oberfläche des Frontends wurde aus zeitlichen Gründen noch nicht implementiert, um das für das Zweitprojekt relevante RL-System zu entwerfen.

## 2. Dokumentation der KI-Methoden für das LTSF

Dieser Abschnitt enthält eine Beschreibung der verwendeten Modelle für das. Im LTSF wird neben dem Modell DLinear [1] die neuere, einfachere Variante NLinear [1] verwendet, die dieselbe Leistung erzielt. Dennoch wird ebenfalls eine eigene Variante namens NDLinear entworfen, die im Gegensatz zu NLinear ein unten beschriebenes Oszillieren der Zeitreihe um die Remainder-Komponente verhindern soll.

### 2.1 Linear und NLinear

Kurz nach dem DLinear Modell wurde die simplere Variante Linear entwickelt, die nun nur noch der Abb. 1 mit einem neuronalen Netzwerk entspricht:

Dabei nutzt Linear ein neuronales Netzwerk, das nur einer einzigen Matrizen-Berechnung entspricht und als Input immer Features vorheriger Zeitreihen-Zeitpunkte erhält. Mit diesem *Look-Back-Window*  $L$  werden in einer nicht-autoregressiven Berechnung gleich mehrere Vorhersagen der zukünftigen *Timesteps*  $T$  mit einem Vorgang berechnet, wodurch die Ausführungszeit von Linear minimal ist und mit einer Grafikkarte unter  $0,4\text{ ms}$  beträgt [1].

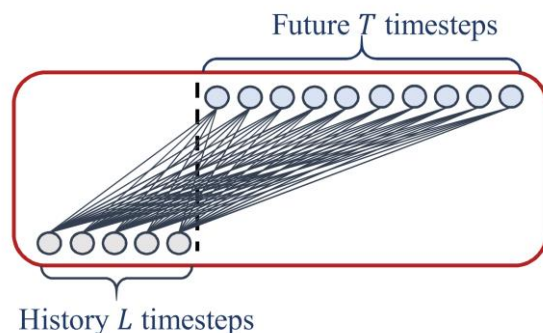


Abb. 1: Eine Darstellung von Linear und NLinear: ein neuronales Netzwerk. [1]

Vorteilhaft von dem Konzept von Linear ist vor allem, dass zeitlich zurückliegende Features separat voneinander miteinbezogen werden und dessen Informationen nicht wie in den

Transformer-Modellen oder LSTM-Modellen in mehreren Berechnungen gewichtet und durch zu viele Transformationen vom Modell z. T. vergessen werden. Stattdessen geht jeder Datenvektor eines Zeitpunktes mit separat vorgenommener und optimierter Gewichtung für jeweils jede einzelne der Vorhersagen ein. Dies verhindert ebenso Folgefehler.

An sich nimmt Linear neben einer *Standard-Scaling* keine Normalisierung der Zeitreihe vor, sodass das Modell Vorhersagen in einem großen Wertebereich der Trend-Komponente vornehmen muss, was es für das Modell beträchtlich schwieriger macht, die gesamte Zeitreihe zu erlernen. Eine Standard-Methode der Zeitreihen-Normalisierung beschreibt dessen Differenzierung, die jedoch die Modell-Leistung massiv einschränkt, da damit die Trend-Informationen zerstört werden. Stattdessen subtrahiert nun die effektivere Variante NLinear den neusten Zeitreihen-Wert vom gesamten Lok-Back-Window, womit dieser Wertebereich verkleinert wird und Trend-Informationen erhalten bleiben. Dieser Normalisierungsvorgang sorgt zudem für eine Minimierung der Unterschiede zwischen Trainings- und Testdaten bzw. zwischen dessen standardisierter Verteilung, was ebenso die Leistung steigert [1].

## 2.2 DLinear und NDLinear

Vor dem NLinear-Modell wurde zuvor das DLinear-Modell entwickelt, das allgemein die gleichen Leistungen wie NLinear erzielt und weiterhin simpel bleibt. Anstelle der Normalisierung in NLinear, trennt Decompose-Linear die Zeitreihe in eine Trend- und Remainder-Komponente wie in Abb. 2 auf, indem es einen bewegten Durchschnitt von der Zeitreihe subtrahiert.

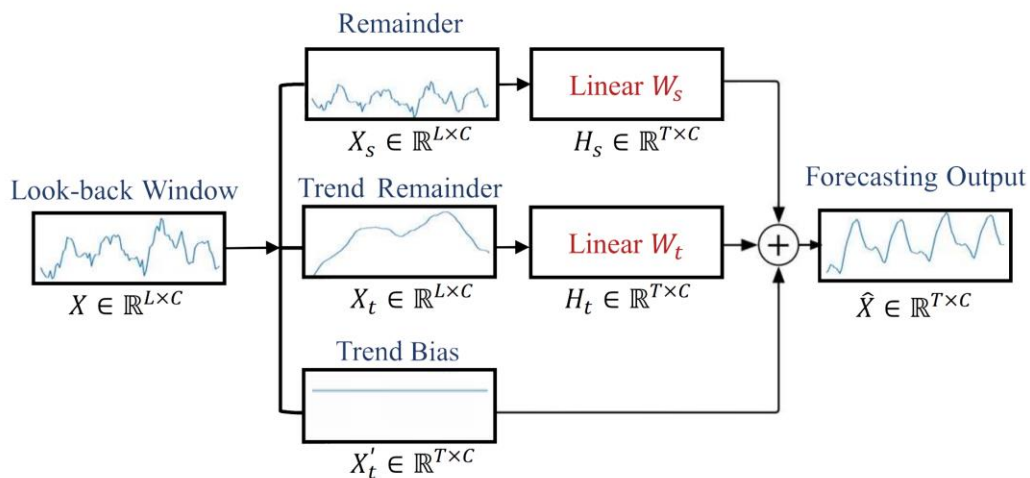


Abb. 2: Die Darstellung des NDLinear-Modells das sich von DLinear lediglich durch einen zusätzlichen Trend-Bias unterscheidet. Die Grafik ist an dem DLinear-Forschungsbericht orientiert [1].

Hiermit ermöglicht DLinear schwankende Remainder-Komponenten von der Trendkomponente zu unterscheiden, indem zwei neuronale Netzwerke separat Vorhersagen für eine der Komponenten treffen, wobei die Vorhersagen zum Schluss wieder zueinander addiert werden.

Schon vor dem Lesen des Modell-Konzepts von NLinear erweist sich in der theoretischen Betrachtung für DLinear der große, sich zeitlich stark ändernde Wertebereich der Trend-Komponente als problematisch für Preisentwicklungs-Vorhersagen zu sein. Dementsprechend wird wie in Abb. 2 mit der Variante NDLinear der neuste Trend-Wert von der Trend-Komponente subtrahiert und als Trend-Bias wieder zu den Vorhersagen dazu addiert.

NLinear und DLinear erzielen fast identische Leistungen [1], während NLinear eine simplere Modell-Struktur besitzt. Der Grund warum genau diese Variante NDLinear im Programm getestet werden

soll, begründet sich dabei bei einem Vergleich dadurch, dass durch Separierung der Trend-Komponente und Remainder-Komponenten die Modell-Leistung gegenüber NLinear gesteigert wird und dadurch, dass auch für DLinear eine Normalisierung wie für NLinear nötig ist.

Hierbei existiert ein weiterer Grund für die Untersuchung des NDLinear-Konzepts: Anstatt die Normalisierung von der gesamten Zeitreihe mit Remainder-Komponente vorzunehmen, wird mit der ausschließlich nötigen Normalisierung der Trend-Komponente das Problem einer oszillierenden Look-Back-Window-Zeitreihe vermieden, die für kompliziertere und schwer erlernbare Bias-Muster sorgt. Dieses Oszillieren kann man sich vereinfacht als eine Sinuskurve vorstellen, die während der Translation auf der Zeitachse zusätzlich für jeden Zeitschritt eine Translation an der Y-Achse vornimmt, sodass diese immer genau mit dem Kurvenverlauf durch den Koordinatenursprung verläuft. Dabei bleibt das Vorhersagezeitfenster in seinem Bias-Wert konstant, während nur das Look-Back-Window für NLinear oszilliert.

### 3. Theorie und Entwurf des RL-Systems

Für den geplanten Entwurf des RL-Systems werden in diesem Abschnitt alle Komponenten beschrieben, die ein RL-System ausmachen. Hierbei wird mit dieser Erklärung der theoretischen Grundlagen der endgültige Entwurf beschrieben.

#### 3.1 Grundlagen des Reinforcement Learnings

Der hauptsächliche Unterschied von RL-Systemen liegt im Gegensatz zu gewöhnlichen KI-Systemen darin, dass ein sogenannter *Agent* bzw. ein RL-Modell mit einer realen oder simulierten Umgebung interagiert. Dieses *Environment* liefert dem *Agent* Daten über deren *State* und ein Feedback in Form von einem *Reward*. Hieraus ergibt sich der zweite wichtige Unterschied zu KI-System: RL-Modelle werden nun anhand eines Belohnungssystems trainiert, das den *Agent* nun ein optimales taktisches und mehrschrittiges Vorgehen antrainiert. Dabei nutzt der *Agent* die *State*- und *Reward*-Daten, um die optimale *Action* für den insgesamt größtmöglichen *Reward* vorherzusagen. All dies wird in der folgenden Abbildung zusammengefasst:

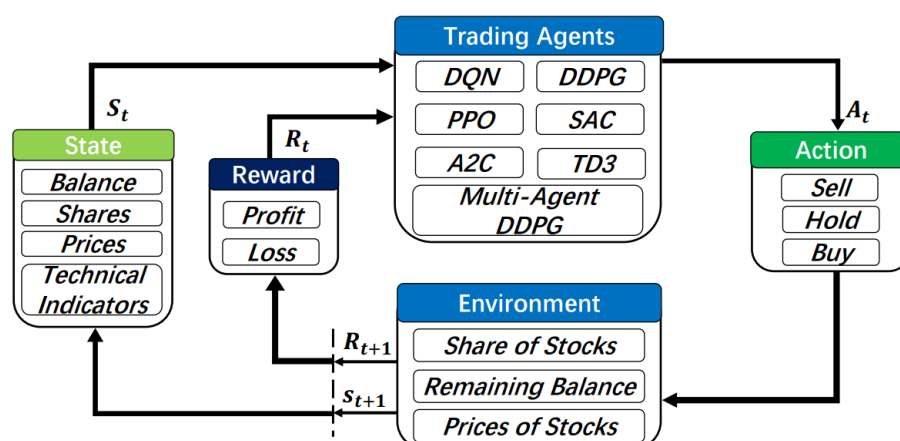


Abb. 3: Die Abbildung eines vollständigen RL-Trading-Systems [7]

Das Training der RL-Modelle findet ähnlich zu den KI-Modellen statt. Wichtig sind hierbei die Begriffe des *Steps* und der *Episode* neben den bekannteren Begriff der Epoche: RL-Modelle sind gezwungen einzelne Schritte bzw. Durchläufe im Environment nach der Abb. 3 durchzuführen, da jetzige States meist von allen vorherigen Aktionen des RL-Modells abhängen und ein Environment nur eine Action je Instanz simuliert. Dabei bezeichnet eine Episode die Gesamtheit aller Steps

sowie deren Sequenz an *States*, *Actions* und *Rewards* bis zum Terminierungs-Zustand der Umgebung. Dagegen bezeichnet die Epoche den Durchlauf einer Episode einschließlich der Modell-Optimierung.

## 3.2 Beschreibung der verwendeten Binance-Umgebung

Alle Experimente werden auf der Plattform Binance wegen deren optimalen Vorraussetzungen für RL-Systeme durchgeführt. Dabei sind wichtige Umgebungsparameter besonders die Kaufs- und Verkaufs-Gebühren mit jeweils 0,1%, die im High-Frequency-Trading maßgeblich den möglichen Endgewinn bestimmen. Eine höhere Gebühr von 0,5% sorgt z.B. dafür, dass ein High-Frequency-Trading gänzlich unrentabel wird, da nur noch Gewinne bei Preissteigerungen von über 1,0 % zu erwarten sind.

Zudem beträgt der Request-Delay bei Daten-Anfragen etwa 250 *ms* innerhalb Europas, die bei Ausführung auf einem Tokio-Server noch auf 20 *ms* reduziert werden können. Crypto-Börsen besitzen keine Tage an denen der Markt geschlossen wird, sodass Modelle simpler angebunden werden können, wobei auch die hohe Liquidität der meisten Crypto-Währungen die nötigen Gewinnchancen im HFT gewährt. Hierbei ist jedoch zu erwähnen, dass eine Frequenz von 20 *ms* nur bei den Währungen mit einem enorm hohen Handelsvolumen wie nur Bitcoin und Ethereum nötig ist. Jede Art der anderen Währungen hat eine deutlich geringere Liquidität, bei welcher die Frequenz von 250 *ms* ausreicht.

Als zusätzliche Referenz für das Betrachten dieser Preisschwankungen wird auf das Live-Dashboard unter <https://www.binance.com/en/markets/overview> verwiesen, das die im Programm verwendeten Daten darstellt. Diese Quelle enthält zudem eine Visualisierung des Order Books mit den Bid- und Ask-Preisen, die für das HFT von Bedeutung sind.

### 3.2.1 State, Action und Reward der Binance Umgebung

#### State-Daten der Umgebung

Relevant bei der Unterscheidung zwischen Umgebungen des HFT- und Scalping-Tradings sind auch deren Charakteristiken ihrer Daten: Die Zeiträume in HFT-Umgebungen sind so klein, dass globale Ereignisse wie Börsen-Crashes so gut wie vernachlässigt werden können, da der *Agent* selbst innerhalb dieser Ereignisse Gewinne aus den stark oszillierenden Preis-Kurven entnehmen kann und einen starken Abfall meist erkennen kann. Hiergegen ist ein Scalping-Trading nicht nur weniger profitabel, sondern auch riskanter durch mögliche Kurs-Einbrüche.

Hierbei liefert die Binance API aktuelle oder historische OHCLV-Daten mit Werten für den Open-, High-, Low- und Close-Preis sowie für das Handelsvolumen. Neben diesen für das Scalping-Trading relevante Daten, können für das HFT die aktuellen Bid- und Ask-Preise des Order Books angefragt werden. Weitere *States* sind Programm-Abhängig und können aus diesen Grunddaten und den *Actions* des RL-Modells gewonnen werden. Für gewöhnlich sind das die Guthaben, die Anzahl an gekauften Anteilen und die technischen Indikatoren.

#### Actions der Umgebung

Die Binance-Umgebung beschränkt dabei die *Actions* des RL-Modells auf das Kaufen, Verkaufen oder Halten von Anteilen, wobei die Menge des gehandelten Volumens für das Kaufen und Verkaufen mit vom *Agent* bestimmt wird. Auch ist die Art der Kaufs- und Verkaufs-

Transaktionstypen im Vorfeld für das RL-System zu wählen. Binance bietet unter anderem *Market*-, *Limit*-, oder *Stop-Orders*, mit der jeweils verschiedene Handels-Strategien umgesetzt werden. *Market-Orders* ist der klassische *Order*-Typ, dessen *Order* bei Anfrage direkt ausgeführt wird. *Limit-Orders* dienen zum Kauf und Verkauf von Anteilen ab gewünschten Preisschwellen, wobei nur andere Marktteilnehmer diese *Order* mit dem Mindestpreis oder einem besseren Preis erfüllen. Dagegen werden *Stop-Orders* ab dem Schwellenpreis direkt wie eine *Market-Order* ausgeführt.

## Reward der Umgebung

Ein numerischer Belohnungswert einer Umgebung wird meist über eine Reward-Funktion definiert, die mit selbstständig definierten Termen die Taktik des RL-Modells maßgeblich bestimmt. In der aktuellen Anwendung gibt es jedoch noch eine flexiblere Variante als eine unvollständige und schwer erlernbare Reward-Funktion zu erlernen, nämlich das sogenannte *Reward-Modeling*, das mithilfe eines zusätzlichen neuronalen Netzwerk den *Reward* tiefgründig anhand von allen relevanten Daten vorhersagen kann.

Hierbei ist der klassische Reward einer Trading-Umgebung hauptsächlich profitorientiert und würde nur wenige Terme als Daten sinnvoll in die Reward-Berechnung mit einbeziehen. Ein neuronales Netz kann effektiver evaluieren, ob eine Investition anhand der momentanen Änderungen von Daten wie den Guthaben, des Gesamtwertes aller Anteile, usw. einen positiven oder negativen Einfluss hat.

## 3.3 Das verwendete PPO-Modell

Im anschließend beschriebenen Entwurf hat sich das Proximal-Policy-Optimization-Modell (PPO) [2] wegen der in der Forschung guten Leistungen und wegen der gleichzeitigen Einfachheit des Modells durchgesetzt. Weitere Vorteile sind der stabile Lernalgorithmus, der selbst bei einem Feature-Engineering nicht zu divergieren droht und im Gegensatz zu anderen Modellen die flexible Anwendbarkeit auf viele Probleme zulässt.

Hierbei gehört PPO zu den Policy-Gradient-Methoden, sodass die *Actions* direkt als Outputs vorhersagt werden. Im Gegensatz zu den vorher bekannten Q-Learning-Methoden, die keine Umgebungs-Aufgaben mit einem stetigen und komplexen Action-Space lösen können, können Policy-Gradient-Methoden diese Probleme mit einzig dem Nachteil der höheren nötigen Datenmenge lösen. Bisherige Policy-Gradient-Methoden dabei besonders vom Nachteil der Stichproben-Ineffizienz betroffen, die das PPO-Modell bei einer simplen Modell-Struktur durch eine einfachere Methode des Gradient-Clippings löst. *Policy*-Gradient-Modelle drohen nämlich durch große Gradienten schnell zu divergieren und von der erlernten Policy  $\pi_\theta(a_t|s_t)$  abzuweichen.

### 3.3.1 Mathematische Beschreibung

Dabei stellt die Policy  $\pi_\theta(a_t|s_t)$  das neuronale Netzwerk des PPO-Modells dar, das ausgehend von den letzten States  $s_t$  und Actions  $a_t$  die Wahrscheinlichkeitsverteilung aller Actions berechnet. Dies wird für diskrete Actions wie folgt notiert  $a_t \sim \pi_\theta(a_t|s_t)$ , wobei  $\theta$  alle Modell-Parameter des Netzwerks darstellt. Im Fall einer stetigen Policy wird dies mit der Notation  $a_t = \mu_\theta(a_t|s_t)$  beschrieben [3]. Zur Optimierung jeder Policy-Gradient-Methode wird hierbei eine sogenannte Advantage-Funktion  $A^\pi(s, a)$  berechnet [3]:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s),$$

Hierbei beschreibt die Q-Funktion  $Q^\pi(s, a)$  die Summe des Rewards  $R(\tau)$  auf der simulierten Trajektorie  $\tau$  des Markow-Entscheidungsprozesses, wenn im betrachteten Zustand  $s$  eine beliebige Action  $a$  ausgeführt und dann der bisherigen Policy  $\pi_\theta$  gefolgt wird. Dagegen beschreibt die Value-Funktion  $V^\pi(s)$  die Summe des Rewards  $R(\tau)$ , wenn schon ab dem betrachteten Zustand  $s$  nur der bisherigen Policy  $\pi_\theta$  gefolgt wird. Somit beschreibt die Advantage-Funktion  $A^\pi(s, a)$  die Differenz des Rewards, wenn abweichend von der Policy  $\pi_\theta$  eine beliebige Action ausgewählt wird. Damit kann die Advantage-Funktion auch wie folgt dargestellt werden:

$$A^\pi(s, a) = r_0 + \gamma V^\pi(s_1) - V^\pi(s_0)$$

Hierbei werden jedoch nicht alle Simulationsschritte durchgeführt und die Q- und V-Funktion bzw. deren Reward-Werte mit z.B. neuronalen Netzwerken approximiert. Hiermit ändert sich für die Notation für  $A^\pi$  zu  $\hat{A}$ . Schließlich wird das Wahrscheinlichkeits-Verhältnis der jetzigen und vorherigen Policy-Verteilung-Wahrscheinlichkeiten benötigt und wie folgt berechnet [2]:

$$r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)}$$

Mit diesem Term wird das folgende Trainings-Ziel in allen PPO-Varianten maximiert [2]:

$$L(\theta) = \mathbb{E}_t[r_t(\theta)\hat{A}_t]$$

Dabei hat sich insgesamt unter allen dieser PPO-Varianten die Variante mit der Funktion  $clip(x, \epsilon) \in [1 - \epsilon; 1 + \epsilon]$  durchgesetzt, die den Wertebereich aller Verhältnisse der Verteilung in  $r_t(\theta)$  einschränkt. Hierbei sorgt weiterhin ein besonders hoher Advantage-Wert für das gezielte Optimieren nach der Policy mit dem insgesamt höchsten Reward [2]:

$$L(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), \epsilon)\hat{A}_t)]$$

### 3.4 Entwurf des RL-Systems

Anhand der erarbeiteten Theorie wird für das 2. Teilprojekt nun das RL-System mit seinen Komponenten entworfen.

#### 3.4.1 Agent

Es wird ein PPO-Modell mit den diskreten Actions des Kaufens, Verkaufens und Haltens festgelegt. Dabei ist es im HFT wichtig kleine Beträge des Order Books genau zu füllen, um nicht zu teureren Preisen einzukaufen. Deswegen wird das Handelsvolumen bei diesen Actions von Kauf oder Verkauf nur in einer profitablen Menge deterministisch berechnet. Dagegen kann im Scalping Trading bei den 3 diskreten Actions stets relativ zum Guthaben gekauft und verkauft werden, da sich der Effekt bei niedrigeren Frequenzen mit längeren Zwischenzeiträumen minimiert.

#### 3.4.2 Environment

Mit Binance wurde ein Großteil des Environments vorgegeben, wobei die Daten-Pipeline für die State-Daten und Reward-Werte entworfen werden muss. Für den Reward wird auf die flexible Lösung des Reward-Modelings beharrt aber für die State-Daten ein relevantes Konzept im Zusammenhang des Feature-Engineerings konzipiert.

Und zwar gibt es die im Code implementierten technischen Indikatoren sowie zusätzlich die verbleibenden 80 Alpha-Faktoren des World-Quant-Papers [4], mit denen experimentiert werden



soll. Dabei gibt es für das HFT neben den 20 unbearbeiteten Order Book Einträgen der profitabelsten Bid- und Ask-Preise sowie deren *Volume*-Werten ein schon implementiertes Feature-Engineering mit etwa 10 Extra-Features.

Neben diesen reinen Environment-Informationen wird eine zweite Art der Agent-Environment-Informationen konzipiert, die für die Entscheidungsfindung des RL-Systems noch eine größere Rolle spielen: Features wie die das aktuelle Guthaben, der aktuelle Reward, die Anzahl der Anteile oder die Änderung des Guthabens geben dem Agent wichtige Informationen über ausschlaggebende Features, die seinen Reward beeinflussen.

### 3.4.3 Netzwerk-Strukturen des Agents

Wegen der potentiell vielen Features bietet sich schnell das Problem des Overfittings und hoher Trainings- und Daten-Berechnungszeiten. Da ebenso die benannten Environment-Informationen teils die Agent-Environment-Informationen überlagern und damit die Entscheidungsfindung des Agents beeinträchtigen erscheint es sinnvoll diese Informationen zu trennen und damit auch ein Overfitting vorzubeugen. Ebenso benötigen kleinere Netzwerke weniger Trainingsdaten.

Insgesamt werden so 3 Strukturen für das Policy-Network vorgeschlagen:

- Eine klassische als  $NN_{\text{classic}}$  neuronale Netzwerk-Struktur mit 4 Schichten. Dabei haben die Hidden-Layer 128 Neuronen. Ein  $NN_{\text{classic}}$ -Netzwerk sollte etwa 25 Features sinnvoll verarbeiten können.
- Eine als  $NN_{\text{split}}$  bezeichnete Struktur mit 2 getrennten  $NN_{\text{classic}}$ -Netzwerken für jeweils die Environment-Informationen sowie die Agent-Environment-Informationen. Hierbei haben diese  $NN_{\text{classic}}$ -Netzwerke 5 Outputs, deren Zwischeninformationen in ein kleines Output-Netzwerk mit 2 Hidden-Layer mit 32 Neuronen geleitet werden.
- Und eine als  $LSTM_{\text{split}}$  bezeichnete Struktur, die zur Informations-Vorverarbeitung die 2 selben  $NN_{\text{classic}}$ -Netzwerke besitzt. Hierbei ist nur das Output-Netzwerk eine LSTM-Zelle, deren Hidden-States für jeden Berechnungsschritt autoregressiv mitgeführt werden. So werden sich Informationsteile der vorherigen Berechnungen gemerkt und effektiv verarbeitet.

### 3.4.4 Bereits codierte Bestandteile des RL-Systems

Anstatt sich bei einem knappen Zeitplan auf das Frontend des Benchmarks-Dashboards zu konzentrieren, wurde die Data-Pipeline des RL-Systems qualitativ implementiert, die die Environment-Daten samt dem Feature-Engineering liefert. Zudem wurden die beschriebenen Klassen der Netzwerk-Strukturen codiert. Neben dem theoretischen Entwurf wird nämlich noch der praktische Entwurf als entscheidend für die Abwicklung des 2. Teilprojektes gehalten.

Dementsprechend wurden diese Bestandteile schon codiert und an die Bibliothek ElegantRL [6] angebunden, dessen RL-Modell-Implementationen leistungsfähiger und effizienter als von den anderen existierenden Bibliotheken FinRL [7] und Stable Baselines3 [8] sind. Die Entwicklung der Datenschnittstellen zwischen Programm und Bibliothek steht aus. Ebenfalls sollten die Environment- und Replay-Buffer-Klassen der Bibliothek für den eigenen Gebrauch noch ausgehend vom vorgestellten Konzept und dem codierten Grob-Entwurf abgewandelt werden.

Die für die spätere Implementation des RL-Systems benötigten Skript-Dateien befinden sich im abgegebenen Code-Verzeichnis, wobei die Dateien der Bibliothek ElegantRL im Verzeichnis *elegantrl\_adapted\_PPO* vorliegen.

### 3.4.5 DLinear-Varianten im RL-System

Mit dem erhaltenen Wissensstand sind zusätzliche LTSF-System innerhalb des RL-Systems nahezu irrelevant und sorgen für eine erhöhte Programm-Komplexität. Somit werden die Konzepte der LTSF-Modelle in der Planung des Zweitprojektes verworfen, da die RL-Systeme das algorithmische Trading komplett selbstständig übernehmen können. Die Erforschung der LTSF-Systeme wird jedoch weitergeführt, da das Vorhersagen von Zeitreihen-Werten an anderen Stellen der Datenanalyse von Bedeutung ist. Deswegen wird der Schwerpunkt ein wenig mehr auf das RL-System gelegt, als im Lastenheft gefordert wird, aber das Programm des Prototyps wird weiterhin als Projekt-Schwerpunkt gesehen.

## 4. Anwendungsbeschreibung

Wie bereits im Lastenheft festgelegt, wird das endgültige Programm die festgelegte Struktur wie in der Abb. 4 haben, wobei im bisherigen Prototyp alle Prozesse der LTSF-Experimente bis auf das Benchmark-Dashboard implementiert wurden. Dabei bieten die jetzigen Programmbestandteile nützliche Funktionen wie die Zusammenstellung und dem Aufnehmen der Datensätze in Echtzeit. Diese Datensätze können mit einem zusätzlichen Feature-Engineering flexibel für andere Zwecke heruntergeladen werden.

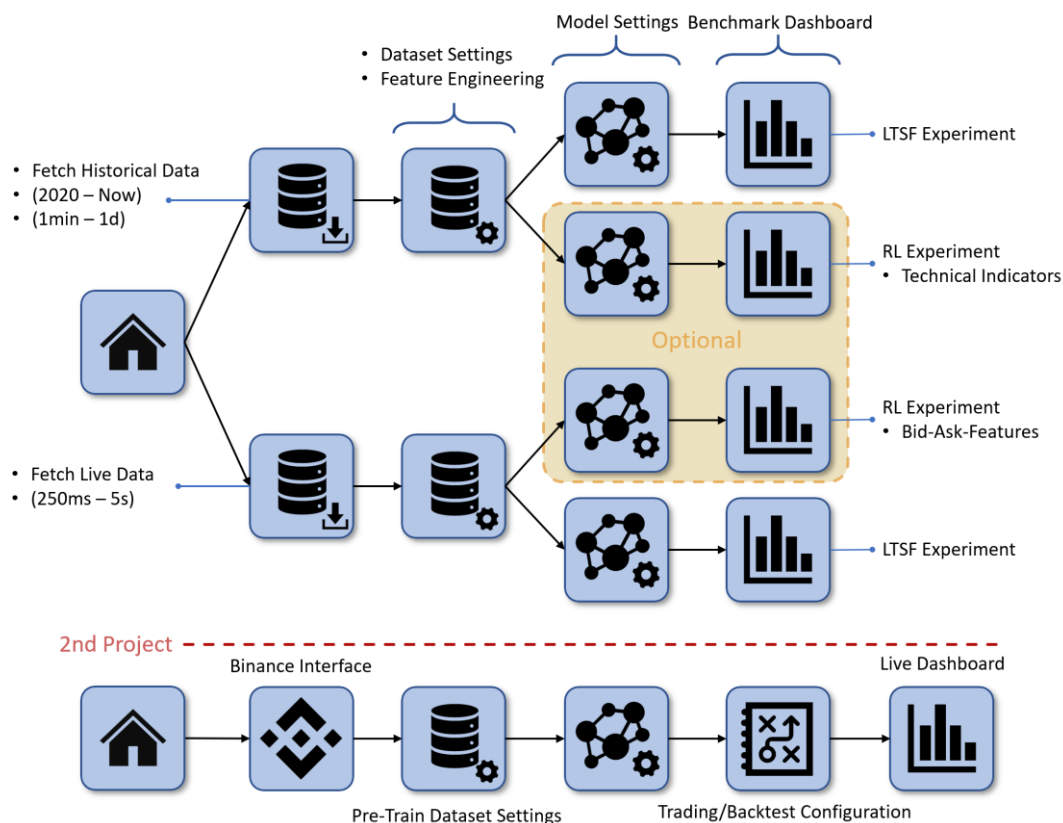


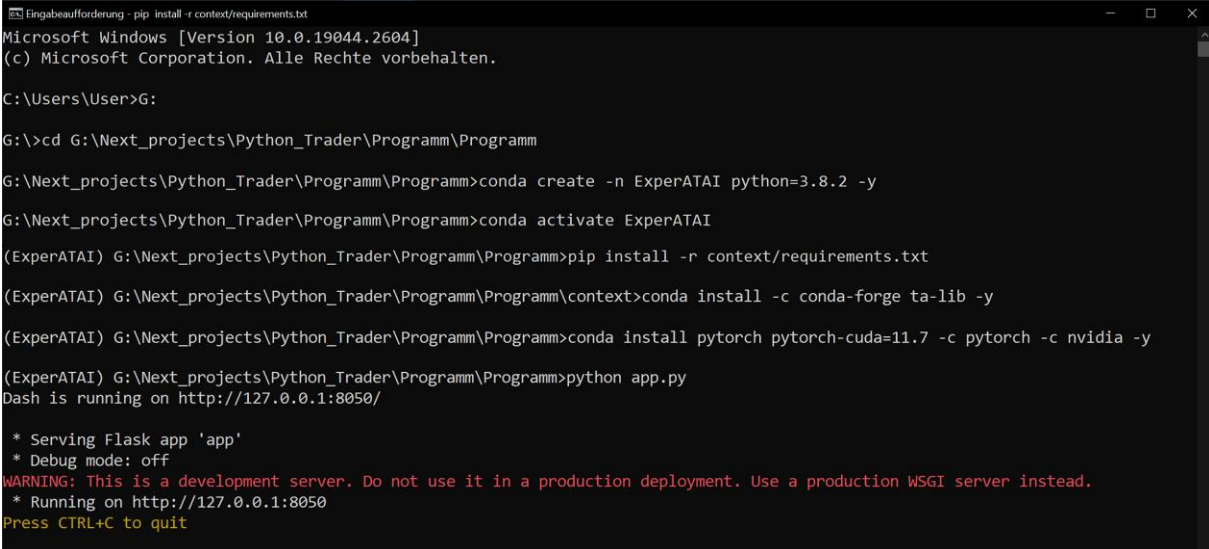
Abb. 4: Die vollständige Darstellung der geplanten Programm-Struktur für sowohl Erst- als auch Zweitprojekt. Von dieser Abbildung wurden die Verlaufsstränge der LTSF-Experimente implementiert.

Daneben existiert die Hauptanwendung des Programms, neuartige KI-Konzepte detailliert für die spätere Anwendung zu untersuchen, wobei das Programm ebenfalls eine Erweiterbarkeit für das Anbinden neuer Konzepte zulässt. Dabei können im jetzigen Programm die aktuellen Varianten

der DLinear-Modelle untersucht werden und bei weiterer Untersuchung relevante Schlüsse für den Bereich des LTSF bezogen auf die Anwendbarkeit getroffen werden. Zur Anwendung des Programms und der Konzept-Entwicklung wurde in der Abb. 8 im Anhang ein BPMN erstellt, das die Programm-Anwendung einschließlich der Konzeptionierung von KI-Systemen verbal beschreibt.

#### 4.1 Installations- und Betriebsanleitung

Zur Installation und Ausführung des Programms wird eine Anaconda-Umgebung benötigt, wobei die gesamte Programm-Ausführung mit dem folgenden Befehlen wie in dieser Abbildung stattfindet:



```
Eingabeaufforderung - pip install -r context/requirements.txt
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\User>G:

G:\>cd G:\Next_projects\Python_Trader\Programm\Programm
G:\Next_projects\Python_Trader\Programm\Programm>conda create -n ExperATAI python=3.8.2 -y
G:\Next_projects\Python_Trader\Programm\Programm>conda activate ExperATAI
(ExperATAI) G:\Next_projects\Python_Trader\Programm\Programm>pip install -r context/requirements.txt
(ExperATAI) G:\Next_projects\Python_Trader\Programm\Programm\context>conda install -c conda-forge ta-lib -y
(ExperATAI) G:\Next_projects\Python_Trader\Programm\Programm>conda install pytorch pytorch-cuda=11.7 -c pytorch -c nvidia -y
(ExperATAI) G:\Next_projects\Python_Trader\Programm\Programm>python app.py
Dash is running on http://127.0.0.1:8050/

* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:8050
Press CTRL+C to quit
```

Abb. 5: Die Auflistung aller Installationsbefehle für das Programm.

In diesem Vorgehen wechselt man dabei in den Programm-Ordner mit dem Python-Skript `app.py`, erstellt die Anaconda-Umgebung, installiert die benötigten Packages. Hierbei ist nur die Installation von *Pytorch-cuda* optional und ermöglicht die Anwendung einer GPU in der Anwendung, sofern die eigene GPU CUDA-fähig ist und die CUDA-Software auf dem eigenen Rechner installiert ist. Nun führt man mit „`python app.py`“ das Main-Skript des Programms aus und erhält die Localhost-Adresse der Webanwendung. Auf die Oberfläche dieser Webanwendung greift man nun über den lokalen Web-Browser zu, indem diese Adresse „`http://127.0.0.1:8050`“ in die URL-Leiste eingefügt wird.

Im Verzeichnis „`Programm\context`“ befindet sich dieselbe Installationsanleitung mit den kopierbaren Setup-Befehlen.

#### 4.2 Gelieferte Programm-Bestandteile

Mit den Skript-Dateien und der Dokumentation werden zusätzlich die verwendeten Datensätze des HFT- und Scalping Trading mitgeliefert. Die Dokumentation wird dabei in analoger sowie digitaler Form abgegeben.

## 5. Dokumentation des Backends

### 5.1 Backend-Struktur

Das sehr umfangreiche Backend des Programms besteht aus mehreren Backend-Kategorien wie den Datensatz-, KI-Modell, Konfigurations-, Experiment-Management- sowie den Benchmark-Klassen, deren volle Dokumentation bei schon etwa 2500 Zeilen Eigenanteil an kompakten Python-Code zu kleinschrittig ist. Mithilfe des Klassendiagramms in Abb. 7 und einer Lupe kann sich hier ein Erster Überblick über die Grobstruktur des Backends verschafft werden.

### 5.2 Datensätze

Innerhalb des Backends werden wie gefordert die zwei Arten der Datensätze durch eigene Skripte zusammengestellt und durch ein breit gefächertes Feature-Engineering verarbeitet, dessen Einfluss mit den genaueren Modell-Testungen untersucht und gegebenenfalls noch verfeinert werden kann. Dabei enthalten die Datensätze des Scalping Tradings OHLCV-Daten aus denen allgemeine Features, technische Indikatoren und testweise die 80 der 101 World-Quant-Features [4] berechnet werden können. Dabei stehen dem Nutzer meist etwa 150 Features zur Verfügung die erleichtert durch etwa 40 Feature-Gruppen über das Frontend ausgewählt werden können. Alle Datensätze des Scalping Tradings werden mittels der abgefragten Daten aus der historischen Binance-Datenbank über ein eigenes Skript zusammengestellt, das über HTTP-Requests an die Webadresse <https://data.binance.vision/> monatliche Datenpakete in Minuten-Frequenz erhält, die korrekt vom Backend in niedrigere Frequenzen überführt werden können.

HFT-Datensätze beinhalten dagegen nur Order-Book-Informationen und nur ein kleines Feature-Engineering, mit welchem hier maximal 30 Features existieren. Dabei wurde zusätzlich zur Zusammenstellung beider Datensätze ein Threading implementiert, das dem Nutzer erlaubt, während der langwierigen Datenaufnahme Experimente durchzuführen. Ebenso wurde viel Aufwand in den Echtzeit-HFT-Data-Fetcher gesteckt, der in regelmäßigen Zeitabständen im Millisekunden-Bereich unter Beachtung von Request-Delays hochqualitative Daten aufnimmt. Dieser Algorithmus kann vereinfacht im Struktogramm der Abb. 9 oder im Quell-Code innerhalb des Threading-Zyklus eingesehen werden.

### 5.3 Modell-Training

Das Modell-Training der implementieren Modell-Varianten Linear, NLinear, DLinear sowie NDLiner findet nach dem gewöhnlichen Datenverarbeitungsvorgang mit Aufteilung in Trainings-, Validierungs- und Testdaten bei gleichzeitiger Standardisierung statt. Hierbei wird das Modell erst mit den über das Frontend festgelegten Parametern trainiert und jede Epoche validiert. Anschließend folgt eine vollständige Testung mit der noch nicht implementierten Benchmark-Klasse, deren ausgegebene interaktive Graphen auf der Benchmark-Oberfläche visualisiert werden würden. Anstelle dieser umfassenden Benchmark werden die Endergebnisse vorerst auf der Konsole mit den Standard-Metriken wie mit dem mittleren absoluten Fehler angegeben.

Hierbei wurde besonders wert auf eine hoch-performante und effiziente Pipeline gelegt, die mit Tensoren anstatt mit Data-Frames arbeitet und auch im Experiment den Arbeitsspeicher-Verbrauch durch einen einmalig instanziierten Datensatz gering hält und langwierige wiederholte Berechnungen vermeidet. Eine Implementierung eines Torch-Dataloaders sorgt dabei für ein

effizientes Batching bei einem gleichzeitigen Daten-Shuffling. Hierfür wird eine zusätzliche Torch-Datensatz-Klasse für das LTSF erstellt, die zueinander gehörende Trainings- und Lösungsdaten als iterierbares Objekt für den Torch-Dataloader zurückgibt.

## 5.4 Experiment-Management-Klassen

Als Methode zur einfachen Erweiterbarkeit benötigt es eine allgemeine Konfigurationsklasse, die die einzelnen Komponenten eines Experiments flexibel initialisiert und alle relevanten Programm-Daten wie Klassen, Datensätze, Modell-Gewichte und Klassen-Parameter mit sich führt. Hierbei sind innerhalb der Klasse *Experiment-Config* zur Übersichtlichkeit alle Parameter aller Experimente gruppiert in Dictionaries untergebracht, auch wenn nicht alle dieser Parameter zwingend benutzt werden. Dabei sind mit der Initialisierung alle Parameter der Klasse *Experiment-Config* auf *None* gesetzt und können mit beliebigen Dictionaries über die Methode *update\_params(param\_dict)* während des Programmdurchlaufs dynamisch gesetzt werden.

Zudem managet diese Klasse die Instanziierung aller Experiment-Bestandteile wie die der KI-Modelle, der Datensätze oder der Benchmarks und hinterlegt diese Klassen mit hohem Arbeitsspeicherverbrauch im Parameter-Dictionary *experiment\_data*. Ebenso werden alle Parameter nur innerhalb der Klasse *Experiment-Config* gesetzt und den erbdenden Experiment-Klassen wie *LTSF-Experiment* oder *RL-Experiment* erweitern die Klasse *Experiment-Config* mit den Methoden *build\_experiment*, *train*, *test* und *validate*.

Wie gefordert, können alle Parameter als YAML-Dateien abgespeichert und ausgelesen werden, womit zukünftige Implementierungen zur schnellen Rekonstruktion durchgeführter Experimente vorgenommen werden kann.

## 6. Dokumentation des Frontends

### 6.1 Grundlegende Funktionen

Im Programm wird die gesamte Oberfläche mit der *Dash*-Klasse im qualitativ hochwertigen Design bei einem kompakten Code umgesetzt. Die interaktiven Designs aller Komponenten wie der Buttons und anderer Schaltflächen haben dabei das elegante Design moderner JavaScript-Komponenten, die mit Python simpel über wenige Zeilen Code spezifiziert werden können. Im Gegensatz zu HTML- oder XML-Dateien gestaltet sich die Webentwicklung damit deutlich übersichtlicher und modularer bei einer direkten Schnittstelle zum Python-Backend. Das Programm als Webapplikation unterstützt somit auch die vom Nutzer schon bekannten Features wie das Hinein- und Herauszoomen bei unterschiedlichen Auflösungen, das automatische Anordnen der Komponenten bei einer Bildschirm-Teilung oder auch das übersichtliche Tab-System des eigenen Browsers, das den Wechsel zwischen mehreren Programm-Oberflächen und anderen Recherche-Webseiten zulässt.

### 6.2 Umsetzung der Oberfläche

Die Oberfläche wurde mit den Bibliotheken Dash und Plotly umgesetzt, wobei Dash die gewöhnlichen Oberflächen Komponenten liefert und Plotly interaktive Graphen zur experimentellen Datenanalyse.

Hierbei ist das Frontend als Mehrfenster-Anwendung umgesetzt und besitzt mehrere Tabs mit unterschiedlichen Oberflächen, die jeweils modular und übersichtlich in einem separaten Python-Skript innerhalb des Ordners *pages* implementiert sind. Dabei besitzt jede Oberfläche eine eigene

Layout-Variable, in der die Oberfläche und deren Komponenten spezifiziert werden. Zusätzlich finden sich in jedem Frontend-Skript die Callback-Methoden der Anwendung, mit denen auch auf das Backend zugegriffen wird.

Hierbei ist das gesamte Frontend über die *Dash*-Klasse implementiert, mit der die Browseranwendung ausgeführt wird. Bei jedem Oberflächenwechsel werden dabei die Layout-Variablen und Callback-Methoden der Klasse überschrieben, womit sich die *Dash*-Klasse über mehrere Python-Skripte erstreckt.

Insgesamt wurden mit 1000 Zeilen Code im Frontend erst die Grundoberfläche mit Programm-interner Dokumentation, das Hauptmenu und die Datensatzkonfiguration mit einer Datenvisualisierung erstellt. Da nur 7 Tage zur Erstellung des Frontends sowie der Dokumentation durch den nicht eingeplanten Wegfall von 3 Wochen Arbeitszeit Verfügung stehen, wird sich hauptsächlich auf die klare Dokumentation aller Konzepte für die spätere Weiterarbeit am Projekt konzentriert und 3 Tage zur Beendigung des Frontends eingeplant.

In den 3 Tagen einschließlich des Wochenendes war es noch möglich die Datenvisualisierung vorzunehmen, womit das Programm zum Einlesen der aufgenommenen Daten und einer Datenanalyse zu gebrauchen ist. Außerdem können mit diesem Frontend die Crypto-Datensätze für andere Aufgaben verarbeitet und heruntergeladen werden, wobei neben der 30 technischen Indikatoren noch etwa 150 weitere Features des Feature-Engineerings berechnet werden können.

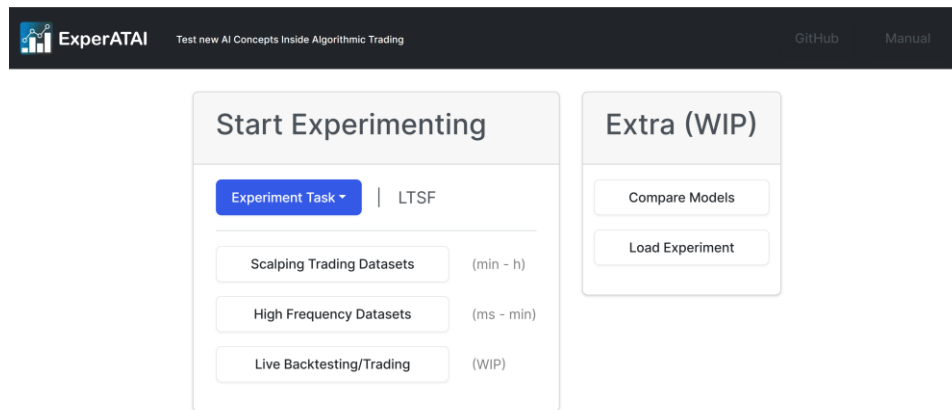
## **6.3 Beschreibung der Programm-Oberflächen**

### **6.3.1 Homepage**

Während der Nutzung des Programms landet der Nutzer zuerst auf der Homepage-Seite, die ein Menü zur Auswahl der Experimente bietet. Hierbei wählt der Nutzer im Programm zwischen dem LTSF bzw. dem HFT aus und wählt seine Art des Datensatzes, die er für diesen Task verwenden möchte, worauf der Nutzer zur Datensatz-Konfiguration geleitet wird.

Alternativ wird das zukünftige Programm einen Echtzeit-Datensatz ausschließlich im HFT unter der Verwendung des finalen aus den Experimenten entnommenen Konzepts zur Verfügung stellen. Ebenso werden im zukünftigen Programm einige Zusatz-Features, wie das Vergleichen mehrerer gespeicherter Modell-Messreihen und das Laden von Experimenten anhand der gespeicherten Parameter-Dateien erfolgen können.

Abb. 6: Die Oberfläche der Frontend-Datei *home.py*.



### 6.3.2 Datensatz-Konfiguration

Nach Auswahl der Datensatz-Kategorie definiert der Nutzer alle relevanten Datensatz-Parameter wie die Input-Features, die Datensatz-Frequenz, die Label-Feature und die Aufteilungs-Anteile in Trainings-, Validierungs- oder Test-Datens. Dabei können je nach der Art des Datensatzes die 64 Features des klassischen Feature-Engineerings, der 30 technischen Indikatoren sowie der 80 WorldQuant-Features ausgewählt werden.

Weitere Funktionen umfassen das Verkleinern des Datensatzes, das Downsampling in längere Frequenzen oder das Herunterladen des instanziierten Datensatz als PKL-Datei. Mit der Schaltfläche „Use Dataset“ gelangt der Nutzer zur Modell-Parameter-Konfiguration und mit der Schaltfläche „Fetch Dataset“ zu einer separaten Oberfläche zum erfassen von Datensätzen. Dabei werden die Nebenparameter die der Währungsname und die ursprüngliche Datensatz-Frequenz aus dem Dateinamen der Datensatz-Datei entnommen.

### 6.3.3 Datensatz-Fetching

Während das Backend des Dataset-Fetchings getestet ist und für die Aufnahme von Datensätzen verwendet werden kann, steht noch die Anbindung an die Oberfläche aus. Mit der vorherigen Auswahl der Task wird dabei die aufzunehmende Datensatzart bestimmt und die relevanten Parameter über die Oberfläche Definiert.

Nach der Bestätigung der Schaltfläche „Fetch Dataset“ wird ein separater Thread gestartet, womit der Nutzer zu einer anderen Tätigkeit übergehen kann oder in der Zeit den Fortschritt der Datensatz-Zusammenstellung überwachen kann.

## 7. Programm-Testungen

Schon während der Programmierung werden Unit-Tests mit den einzelnen Klassen durchgeführt, da eine spätere Testung zusammen mit den Frontend-Festlegungen zu aufwendig wäre. Nach der Implementation des Backends folgt schließlich ein vollständiger System-Test für das anschließend implementierte Frontend, um grobe Fehler zu beheben.

Zusätzlich wird ein Installationstest der Software auf einem anderen Windows-Rechner bei einem kurzen wiederholten System-Test durchgeführt. Dabei benötigt die Installation der Anwendung

Internet-Zugang ohne Admin-Zugang. Jedoch kann die Software nicht an Rechnern installiert werden, deren Internetzugang maßgeblich durch einen Proxy-Server eingeschränkt ist, der die Ausführung der meisten Anaconda-Befehle verhindert. Schließlich muss noch ein Performance-Test für die KI-Modelle Linear, NLinear, DLinear und NDLinear vorgenommen werden, um fehlerhafte Implementationen auszuschließen.

## 8. Erfüllung des Anforderungskatalogs

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.	✓ (Schwerpunkt)
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.	(55%) (Implementierung des Backends 90%) (keine vollständige Benchmark) (keine Modell- Testergebnisse)



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.

*(keine Oberflächen  
für Modell-Training  
und Datensatz-  
Zusammenstellung)*

✓

✓

## 9. Fazit

Trotz der einzig fehlenden Oberflächen für Benchmark und Modell-Training legt die Erfüllung des Projekts den verlässlichen Grundstein zur weiteren Durchführung der folgenden Teilprojekte. Dabei wurden mit diesem Programm schon Pipelines für die endgültigen Algorithmen implementiert und einige Konzepte wie die Linear-Modelle für die finale Umsetzung ausgeschlossen, aber für andere Prozesse nutzbar gemacht. Denn mit dem Entwurf des RL-Systems wird offenbart, dass ein RL-System eigenständig und simpler ohne Zeitreihen-Vorhersagen agieren kann. Besonders die Erweiterbarkeit und Modularität mit der Implementierung eigener Klassen hilft das Programm auch bei zukünftigen zu erforschenden KI-Konzepten für schnelle Experimente zu erweitern.

Dabei ist es sehr bedauerlich, dass einfach nicht genügend Zeit zur Erfüllung des Projektes verblieb und am Ende zwar die Anforderungen für ein Schulprogramm erfüllt wurden, aber die Anforderungen des Lastenheftes nur etwa zur Hälfte. Innerhalb des abgegebenen Programms mit etwa 1500 Zeilen Code an involvierten Backend-Skripten und der 1000 Zeilen Code des Frontends, kann durchaus schon mit dem abgegebenen kleinen Programm-Teil eine relevante Funktionalität der Datensatz-Analyse und -Verarbeitung erzielt werden.

### 9.1 Weitere Planung

Mit dem Abschluss des Erstprojekts steht die Vervollständigung des Erstprojektes und die Durchführung des nächsten Teilprojektes an. Dabei wird entschieden, den verbleibenden Projekt-Anteil in ein Zweit-Projekt und ein Drittprojekt aufzuteilen. Im Zweitprojekt werden Programm und Oberfläche weiter vervollständigt und das entworfene RL-System implementiert. Zudem werden mit dem Zweitprojekt sowohl das LTSF- als auch das RL-System für die praktische Anwendung finalisiert und ein Vortrainings-Konzept für das Drittprojekt beschrieben.

Im Drittprojekt soll die Schnittstelle zu Binance mit den nötigen Oberflächen erstellt werden, um Echtzeittests mit realen Binance-State-Daten zu erhalten, die abhängig von den Actions des RL-Systems ausgegeben werden, jedoch zur Simulation kleine Echtgeld-Beträge von mindestens 15,00€ benötigen. Hierbei soll das Programm an die aktuellen Industrie-Standards angepasst werden und dazu an einen Server in Tokyo angebunden werden. Dies erlaubt schneller Daten im Frequenz-Bereich von 20 ms aufzunehmen und das aktuelle Programm für die Anwendung von verschiedenen GPU- und TPU-Systemen testen zu können.

Ziel dabei ist nicht unbedingt später Gewinne mit der Anwendung zu erhalten, sondern die Entwicklung von Programmen zur Datenanalyse mit Dash zu üben und die praktische Anwendung

von RL-Methoden zu erproben. Die möglichen Gewinne nach einem eingeplanten Entwicklungszeitraum von einem Jahr können zur Finanzierung der folgenden Server-Experimente und auch späterer Projekte genutzt werden, um diese maßgeblich zu unterstützen.

## Literaturverzeichnis

- [1] A. Zeng, M. Chen, L. Zhang und Q. Xu, „Are Transformers Effective for Time Series Forecasting?“, arXiv:2205.13504v2, 2022.
- [2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford und O. Klimov, „Proximal Policy Optimization Algorithms“, arXiv:1707.06347, 2017.
- [3] OpenAI, „OpenAI Spinning Up“, 2018. [Online]. Available: [https://spinningup.openai.com/en/latest/spinningup/rl\\_intro.html](https://spinningup.openai.com/en/latest/spinningup/rl_intro.html). [Zugriff am 07.03.2023].
- [4] Z. Kakushadze, „101 Formulaic Alphas“, arXiv:1601.00991v3, 2016.
- [5] 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.
- [6] AI4Finance, „ElegantRL “小雅”: Massively Parallel Library for Cloud-native Deep Reinforcement Learning“, [Online]. Available: <https://github.com/AI4Finance-Foundation/ElegantRL>. [Zugriff am 08.03.2023].
- [7] 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.
- [8] A. Hill, „Stable Baselines3“, [Online]. Available: <https://github.com/DLR-RM/stable-baselines3/>.
- [9] 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.
- [10] 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](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3690996), 2020.
- [11] S. Sun, R. Wang und B. An, „Reinforcement Learning for Quantitative Trading“, arXiv:2109.13851, 2021.
- [12] 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.

- [13] X.-Y. Liu, H. Yang, J. Gao und C. D. Wang, „FinRL: Deep Reinforcement Learning Framework to Automate Trading in Quantitative Finance,“ arXiv:2111.09395v1, 2021.

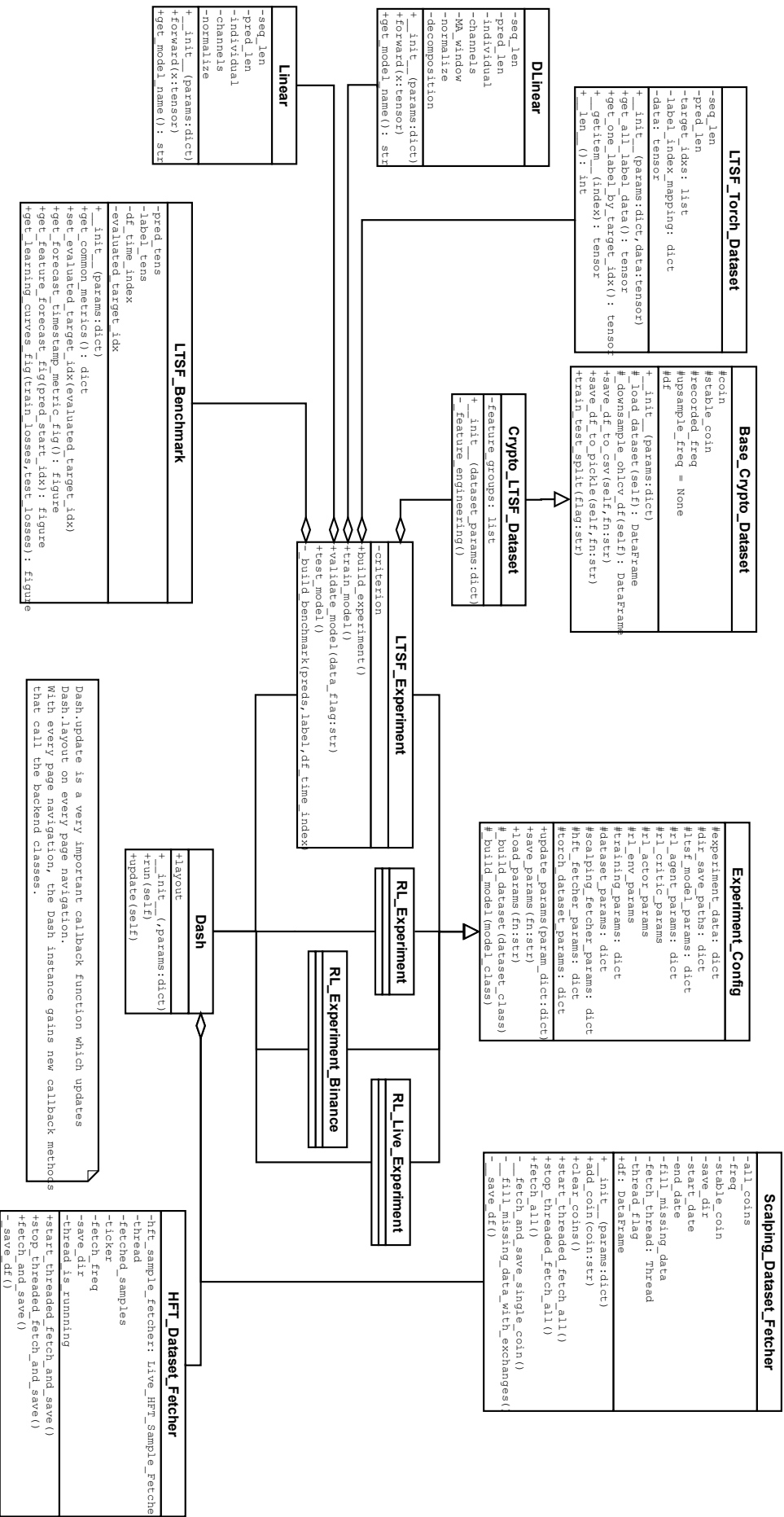


Abb. 7: Das Klassendiagramm des Programms mit Auflistung der wichtigsten Klassen. Einige irrelevante Parameter wurden zur Übersicht weggelassen. Alle Multiplizitäten stellen 1-1-Beziehungen dar und werden ebenfalls weggelassen und in dieser Abbildung die Klassen der nicht vollständig implementierten RL-Experiment-Klassen nicht aufgeführt.





Abb. 9: Das Struktogramm der Datensatz-Zusammenstellung für das High-Frequency-Trading. Hierbei werden zu festen Frequenz-Zeitpunkten im Millisekunden-Bereich abgefragt und die verarbeiteten Abfrage-Daten in einem Python-Dictionary zurückgegeben.