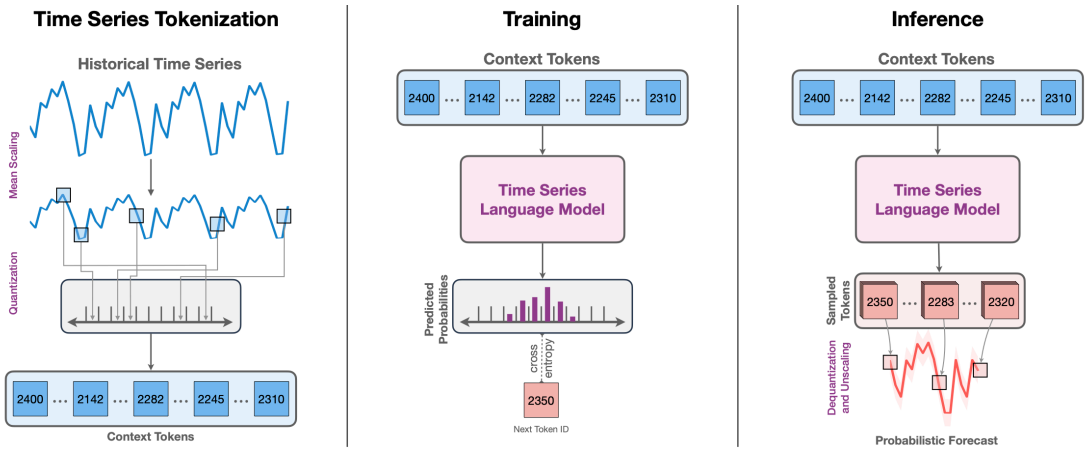
Chronos: Learning the Language of Time Series

מתן לבינטוב יובל שוורץ גבריאל מגידוב שי קרנצברג

1 תיאור כללי של הרעיון

אנו לוקחים מודל מאומן מראש של Amazon שהוא ממשפחת pretrained time series forecasting models בשם Chronos. Chronos הוא מודל חיזוי של סדרות עתיות (Time Series) שמבוסס על ארכיטקטורה של LLM. המודל לוקח סדרה עיתית, עושה עליה טרנספורמציות של נתונים כדי להפוך אותה ל context tokens שאותם הוא שולח למודל השפה שספציפית אומן על סדרות עיתיות. לאחר שמודל השפה קיבל את הcontext, אנו מבקשים ממנו לחזות את ה token הבא שאותו המודל מחזיר לפורמט המקורי של הסדרה העיתית.



איור 1: הארכיטקטורה של Chronos

באמצעות Chronos ונתוני עבר של Bitcoin ו Ethereum אנו מנסים לחזות את מחירי הסגירה של Bitcoin בנר הבא. מצאנו שהמודל חוזה בצורה אמינה יותר ככל שאנו מביאים לו יותר נתוני עבר רלוונטים וכדי להשתמש ביתרון של Bitcoin, שהוא נסחר 24/7 וגם שקל להשיג ממנו נתונים תוך יומיים מאוד מפורטים, בחרנו לעבוד עם נרות שעתיים.
בבדיקות שערכנו, מצאנו שיש קורלציה חזקה בין מחירי Bitcoin למחירי Ethereum, ולכן החלטנו לנסות להשתמש בשני הנתונים יחדיו ומצאנו שאמינות המודל גדלה.
בנוסף, בחרנו להתשמש באינדיקטור טכני STC - Schaff Trend Cycle Indicator על מנת לסנן את התחזיות של המודל. STC הוא אינדיקטור טכני המשלב בין ממוצעים נעים למחזורי זמן כדי לזהות מגמות בשוק בצורה מהירה ומדויקת.

2 פירוט מקורות נתונים

את הנתונים לקחנו מ Binance. השתמשנו ב API שלהם ובספריית requests, ניתן למצוא את הקוד ב main.py תחת הפונקציה get_binance_historical_data(·). את הבסיס לפונקציה לקחנו מהמעבדה עם יונתן והתאמנו אותה לצרכים שלנו על ידי כך שהוספנו את האופציה לתאריך סיום. הנתונים הם. לקחנו נתוני עבר שעתיים של Bitcoin ו Ethereum מסוג OHLCV.

```python
import pandas as pd
import requests

def make_api_call(base_url, endpoint="", method="GET", **kwargs):
    # Construct the full URL
    full_url = f'{base_url}{endpoint}'

    # Make the API call
    response = requests.request(method=method, url=full_url, **kwargs)

    # Check if the request was successful (status code 200)
    if response.status_code == 200:
        return response
    else:
        # If the request was not successful, raise an exception with the error message
        raise Exception(f'API request failed with status code {response.status_code}: {response.text}')

def get_binance_historical_data(symbol, interval, start_date):

    # define basic parameters for call
    base_url = 'https://fapi.binance.com'
    endpoint = '/fapi/v1/klines'
    method = 'GET'

    # Set the start time parameter in the params dictionary
    params = {
        'symbol': symbol,
        'interval': interval,
        'limit': 1500,
        'startTime': start_date # Start time in milliseconds
    }


    # Make initial API call to get candles
    response = make_api_call(base_url, endpoint=endpoint, method=method, params=params)

    candles_data = []

    while len(response.json()) > 0:
        # Append the received candles to the list
        candles_data.extend(response.json())

        # Update the start time for the next API call
        params['startTime'] = candles_data[-1][0] + 1 # last candle open_time + 1ms

        # Make the next API call
        response = make_api_call(base_url, endpoint=endpoint, method=method, params=params)


    # Wrap the candles data as a pandas DataFrame
    columns = ['open_time', 'open', 'high', 'low', 'close', 'volume', 'close_time', 'quote_asset_volume',
               'number_of_trades', 'taker_buy_base_asset_volume', 'taker_buy_quote_asset_volume', 'ignore']
    dtype={
    'open_time': 'datetime64[ms, Asia/Jerusalem]',
    'open': 'float64',
    'high': 'float64',
    'low': 'float64',
    'close': 'float64',
    'volume': 'float64',
    'close_time': 'datetime64[ms, Asia/Jerusalem]',
    'quote_asset_volume': 'float64',
    'number_of_trades': 'int64',
    'taker_buy_base_asset_volume': 'float64',
    'taker_buy_quote_asset_volume': 'float64',
    'ignore': 'float64'
    }

    df = pd.DataFrame(candles_data, columns=columns)
    df = df.astype(dtype)

    return df
```

3 סטטיסטיקה תאורית של הנתונים וכן הסבר על תהליכי ניקיון וטיוב של הנתונים

3.1 סטטיסטיקה תאורית

```
df_btc.describe()
```

✓ 0.0s

	open	high	low	close	volume
count	8809.000000	8809.000000	8809.000000	8809.000000	8809.000000
mean	44415.099205	44571.368101	44253.585015	44419.749415	12770.593399
std	15948.579328	16030.119080	15859.237571	15949.054301	15395.367754
min	25008.200000	25125.000000	24581.000000	25008.300000	736.925000
25%	29521.100000	29596.900000	29450.200000	29523.700000	4895.378000
50%	41804.800000	41922.500000	41679.000000	41805.200000	7989.908000
75%	62853.700000	63139.000000	62549.200000	62864.000000	14768.874000
max	73644.500000	73881.400000	73270.200000	73644.600000	355275.447000

איור 2: סטטיסטיקה תאורית של Bitcoin

```
df_eth.describe()
```

✓ 0.0s

	open	high	low	close	volume
count	8809.000000	8809.000000	8809.000000	8809.000000	8.809000e+03
mean	2434.813695	2444.444616	2424.489079	2435.032094	1.042680e+05
std	739.999738	745.005093	734.482716	740.057992	1.115840e+05
min	1528.170000	1530.760000	1492.660000	1528.180000	7.687973e+03
25%	1837.750000	1842.090000	1834.040000	1837.900000	4.386083e+04
50%	2233.830000	2241.940000	2223.580000	2233.830000	7.116939e+04
75%	3084.980000	3099.270000	3069.240000	3085.080000	1.214815e+05
max	4072.930000	4098.640000	4066.800000	4072.920000	1.759250e+06

איור 3: סטטיסטיקה תאורית של Ethereum

3.2 תהליך הניקוי והתיקון

בקובץ config.json ניתן לבחור אם המשתמש מעוניין לבדוק ולנקות את הדאטא, כמובן שזה מומלץ ואף חיוני. פונקציית תיקון וניקוי הדאטא עוברת על הדאטא ומחפשת ערכי open / close שנמצאים מחוץ לטווח low / high שלהם. כשהיא מוצאת כאלו היא מוחקת את הערך (היא שמה NaN במקום אותו ערך). לאחר מכן קוראים לפונקציה, mean_imputation_without_leakage(·) שהיא מחליפה את הערכים החסרים בממוצע הערכים הקודמים אשר עומדים בטווח (בטווח הכוונה לטווח low / high).

```python
def mean_imputation_without_leakage(srs, high, low):
    filled_srs = srs.copy()
    for index in range(len(srs)):
        if pd.isnull(srs.iloc[index]):
            valid_values = srs.where((srs >= low[index]) & (srs <= high[index])).dropna()
            if len(valid_values) > 0:
                filled_srs.iloc[index] = valid_values.expanding().mean().iloc[-1]
            else:
                filled_srs.iloc[index] = np.nan  # In case there are no valid values to compute the mean
    return filled_srs

def clean_and_fill_data(data):
    # Check and replace bad values with NaN
    bad_open_condition = (data['open'] < data['low']) | (data['open'] > data['high'])
    bad_close_condition = (data['close'] < data['low']) | (data['close'] > data['high'])

    data.loc[bad_open_condition, 'open'] = np.nan
    data.loc[bad_close_condition, 'close'] = np.nan

    # Check for NaN values in 'volume' and replace with NaN (if not already NaN)
    data['volume'] = data['volume'].replace(0, np.nan)

    # Fill NaN values using the mean_imputation_without_leakage function
    data['open'] = mean_imputation_without_leakage(data['open'], data['high'], data['low'])
    data['close'] = mean_imputation_without_leakage(data['close'], data['high'], data['low'])
    data['volume'] = data['volume'].fillna(method='ffill').fillna(method='bfill')

    return data
```

איור 4: תהליך ניקוי ותיקון הנתונים

4 תיאור מפורט של האסטרטגיה

בכל נקודת זמן, אנחנו שולחים context ל Chronos באורך של 1000 נרות, היות ואנחנו משתמשים בנרות שעתיים, זה בערך יוצא 41 ימים של נתונים. ה context מורכב מהעמודות הבאות :

- btc_close - מחירי הסגירה של Bitcoin בנר
- btc_high - המחיר הגבוה ביותר של Bitcoin בנר
- btc_low - המחיר הנמוך ביותר של Bitcoin בנר
- btc_volume - כמות המסחר של Bitcoin בנר
- eth_close - מחירי הסגירה של Ethereum בנר
- eth_volume - כמות המסחר של Ethereum בנר

היות וChronos הוא מודל פרופבליסטי (לא דטרמיניסטי) אז יש חשש שנקבל ניבוי קיצוני מתוך התפלגות האפשרית, לכן אנחנו שולפים 20 דגימות מתוך התפלגות ולוקחים את החציון של התוצאות שקיבלנו. או במילים יותר פשוטות, אנחנו מנבים את המחיר הבא 20 פעמים ולוקחים את החציון של הערכים שקיבלנו. על מנת לשפר את האסטרטגיה ולא להיות תלויים לגמרי במודל, החלטנו גם להוסיף את האינדיקטור הטכני STC על מנת לנבא בצורה טובה יותר את המגמות בשוק.
האינדיקטור הטכני STC (Schaff Trend Cycle) משמש לזיהוי מגמות בשוק המסחר. הוא משלב את ממוצע התנועה האקספוננציאלי (EMA) ואת האינדיקטור MACD[1] למתן אותות קנייה ומכירה בצורה מהירה ומדויקת יותר. ה-STC מתחשב גם במגמות ארוכות טווח וגם במגמות קצרות טווח, מה שמאפשר לזהות שינויים במגמה בשלב מוקדם יותר מאינדיקטורים מסורתיים. בפרויקט שלנו, בחרנו להשתמש בפרמטרים הבאים לאינדיקטור: אורך STC של 80, תקופת EMA קצרה של 27, תקופת EMA ארוכה של 50, מקדם החלקה של 0.5, ערך רמת מכירת יתר של 70 וערך רמת קניית יתר של 30.

כלל קנייה וכלל מכירה

כלל הקנייה שלנו הוא כדילקמן: אם החציון של התפלגות תחזיות גדול מהמחיר עכשיו (Bitcoin Close Price) כלומר הוא חוזה עליה ו ה STC קטן מערך רמת קניית היתר, כלומר ה STC חושב שכדאי לקנות, אז האסטרטגיה שלנו תייצר איתות קנייה.
כלל המכירה שלנו הוא כדילקמן: אם החציון של התפלגות תחזיות קטן מהמחיר עכשיו (Bitcoin Close Price) כלומר הוא חוזה ירידה ו ה STC גדול מערך רמת מכירת היתר, כלומר ה STC חושב שכדאי למכור, אז האסטרטגיה שלנו תייצר איתות מכירה.
נכון לכרגע, היישום של Chronos שונה בין מחשבי Mac M chips למחשבי Windows with Nvidia GPU. לכן אנחנו בודקים מהי מערכת הפעלה שמורץ עליו הקוד.

היישום למחשבי Mac M chips

ל Chronos יש branch נסיוני למעבדי ARM של מק, אנחנו גם מייבאים קוד מ - branch שונה. והשוני הנוסף היחיד הוא שאת ה context אנחנו שולחים למודל בתור np.array.

היישום למחשבי Windows with Nvidia GPU

בגרסה הזאת אנחנו משתמשים בגרסה הרגילה של Chronos ושולחים את ה context בתור torch.tensor. ומחליטים איפה להריץ את המודל, האם ב CPU או ב GPU.

[1] MACD (Moving Average Convergence Divergence) הוא אינדיקטור טכני שמשמש לזיהוי שינויים במומנטום של מניות ונכסים אחרים, על ידי השוואת שני ממוצעים נעים אקספוננציאליים (EMA) - אחד קצר טווח ואחד ארוך טווח. האותות הנוצרים מההבדלים ביניהם יכולים להצביע על הזדמנויות קנייה ומכירה.

5 Backtesting

את הבסיס לקוד לקחנו מהעבודה הסופית של המעבדה מסמסטר קודם.
יצרנו קובץ config.json שבעזרתו אפשר לראות ולשלוט על כל הקבועים שמשפיעים גם על האסטרטגיה וגם על ה Backtesting. בקובץ ניתן לשלוט על תאריך ההתחלה, תאריך הסיום, stop_loss, comission, slippage ועוד.
שימו לב שמחליטים רק על ה stop_loss, ה-take_profit תמיד יהיה פי 2 מה -stop_loss.

```
{
  "backtest": {
    "symbol": "BTCUSDT",
    "context_symbol": "ETHUSDT",
    "interval": "1h",
    "start_date": { "year": 2023, "month": 6, "day": 16 },
    "end_date": { "year": 2024, "month": 6, "day": 17 },
    "window": 1000,
    "balance": 100000,
    "save_to_csv": true,
    "clean_the_data": true,
    "slippage_factor": 20,
    "comission": 5,
    "sl_rate": 0.05
  },

  "stc": {
    "stc_length": 80,
    "fast_length": 27,
    "slow_length": 50,
    "aaa": 0.5,
    "over_sold": 70,
    "over_bought": 30
  },

  "chronos": {
    "prediction_length": 1,
    "num_samples": 20,
    "temperature": 1.0,
    "top_k": 50,
    "top_p": 1.0
  }
}
```

איור 5: קובץ config.json

על מנת להריץ את הBacktesting יש צורך להריץ את הקובץ main.py.
התוכנית תתחיל בלהוריד את הדאטא בטווח הזמן שנבחר, לאחר מכן היא תבדוק, תנקה ותתקן את דאטא (במידה וזה

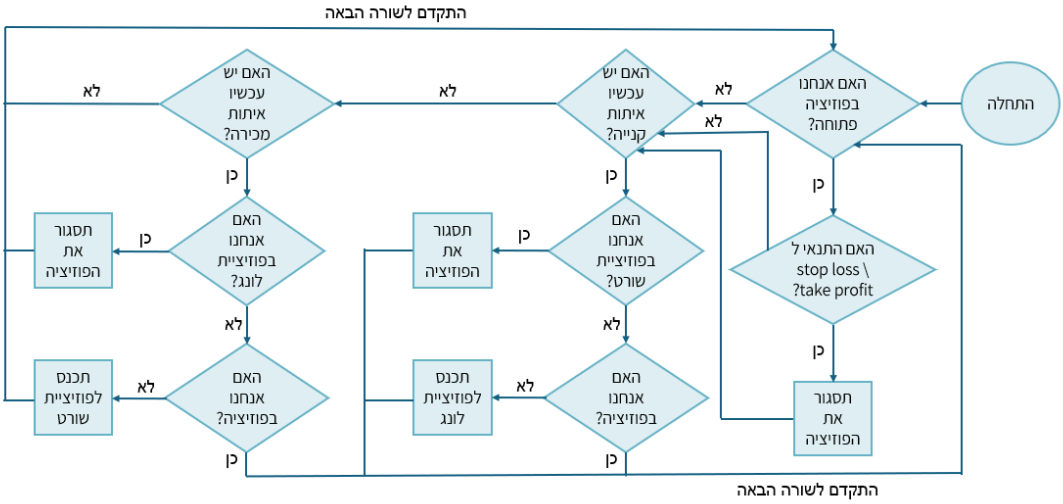
התבקש על ידי המשתמש בקובץ config.json). לאחר מכן התוכנית תחשב את הסיגנלים לפי האסטרטגיה והיא תיצור 2 אסטרטגיות בסיסיות (Buy and Hold, Sell and Hold) שיהיה לנו בסיס להשוואה. לאחר שיש לנו את הסיגנלים, נקרא לפונקציה $backtest(\cdot)$. הפונקציה $backtest(\cdot)$ תסמלץ עבורנו את המסחר ולאחר מכן נקרא לפונקציה $evaluate\_strategy(\cdot)$ על מנת שהיא תחשב את הרווח ומדדי הביצוע של האסטרטגיה.

```python
context = [data.high.values, data.low.values, context_df.close.values, data.volume.values, context_df.volume.values]
    strategy = ChronosStrategy(window)
    data["strategy_signal"] = strategy.calc_signal(data.copy(deep=True),context)

    BAH_strategy = BuyAndHoldStrategy(WINDOW=window)
    SAH_strategy = SellAndHoldStrategy(WINDOW=window)
    data["BAH_signal"] = BAH_strategy.calc_signal(data.copy(deep=True))
    data["SAH_signal"] = SAH_strategy.calc_signal(data.copy(deep=True))

    backtest_df = backtest(data=data.copy(deep=True), strategy=strategy, signals=data["strategy_signal"],
                    starting_balance=balance, slippage_factor=slippage_factor, comission=comission, sl_rate=sl_rate)
    print("\nPortfolio value:", backtest_df["portfolio_value"].iloc[-1], "\n")
    evaluate_strategy(backtest_df, "Chronos Strategy")
    BAH_backtest_df = backtest(data=data.copy(deep=True), strategy=BAH_strategy, signals=data["BAH_signal"],
                    starting_balance=balance, slippage_factor=slippage_factor, comission=comission, sl_rate=None)
    print("\nPortfolio value:", BAH_backtest_df["portfolio_value"].iloc[-1], "\n")
    evaluate_strategy(BAH_backtest_df, "Buy and Hold Strategy")
    SAH_backtest_df = backtest(data=data.copy(deep=True), strategy=SAH_strategy, signals=data["SAH_signal"],
                    starting_balance=balance, slippage_factor=slippage_factor, comission=comission, sl_rate=None)
    print("\nPortfolio value:", SAH_backtest_df["portfolio_value"].iloc[-1], "\n")
    evaluate_strategy(SAH_backtest_df, "Sell and Hold Strategy")
```

הפונקציה $backtest(\cdot)$ תסמלץ מסחר בכך שהיא תעבור שורה שורה על הדאטא. בכל שורה, אם אנחנו בפוזיציה התוכנית תבדוק אם התנאי לStop loss / Take profit התקיים ואם כן היא תסגור את הפוזיציה בהתאם למחירים. לא משנה אם אנחנו בפוזיציה או לא, היא תבדוק את האיתות לאותה שורה, אם יש איתות קנייה אז היא תבדוק אם אנחנו בפוזיציית short ואם כן היא תסגור את הפוזיציה, אם לא היא תבדוק אם אנחנו בכלל בפוזיטיציה, אם כן אז לא נעשה דבר, ואם לא תיכנס לפוזיציית long. אם יש איתות מכירה, אז מתקיים ההפך. כשהלולאה מגיעה לשורה האחרונה בדאטא, אם אנחנו בפוזיצייה היא סוגרת אותה.



איור 6: תרשים זרימה של הBacktesting

ב backtest יש גם קריאה לפונקציית $calc\_realistic\_price(\cdot)$ מתוך ההבנה שכל פעם שנרצה לעשות עסקה, לא בהכרח נתפוס אותה במחיר שרצינו וגם צריך לקחת בחשבון את העמלות מסחר. כפי שאפשר לראות בקובץ config.json בחרנו $slippage\_factor$ של 20, עמלה של 5 דולר לכל מכירה וקנייה. (יש משתנה שסופר את כמות העסקאות ומוריד בסוף תקופת המסחר backtesting את העמלות מהרווח).
בקובץ config.json יש אפשרות לבחור אם לשמור את הדאטא והתוצאות לקובץ csv.

6 תוצאות

לאחר הBacktesting אנו קוראים לפונקציה $evaluate\_strategy(\cdot)$ לכל אחת מהאסטרטגיות, הפונקציה מחשבת את מדדי הביצוע הבאים :

- Sharpe Ratio
- Sortino Ratio
- Max Drawdown
- Annualized Return
- Calmar Ratio

```
Number of Trades : 18

Portfolio value: 444235.6134478221


Results for Chronos Strategy:
Total Return: 345.04%
Annualized Return: 433.78%
Annualized Sharpe Ratio: 9.25
Sortino Ratio: 12.44
Max Drawdown: 20.21%
Calmar Ratio: 21.46
```

(א) ביצועי Chronos

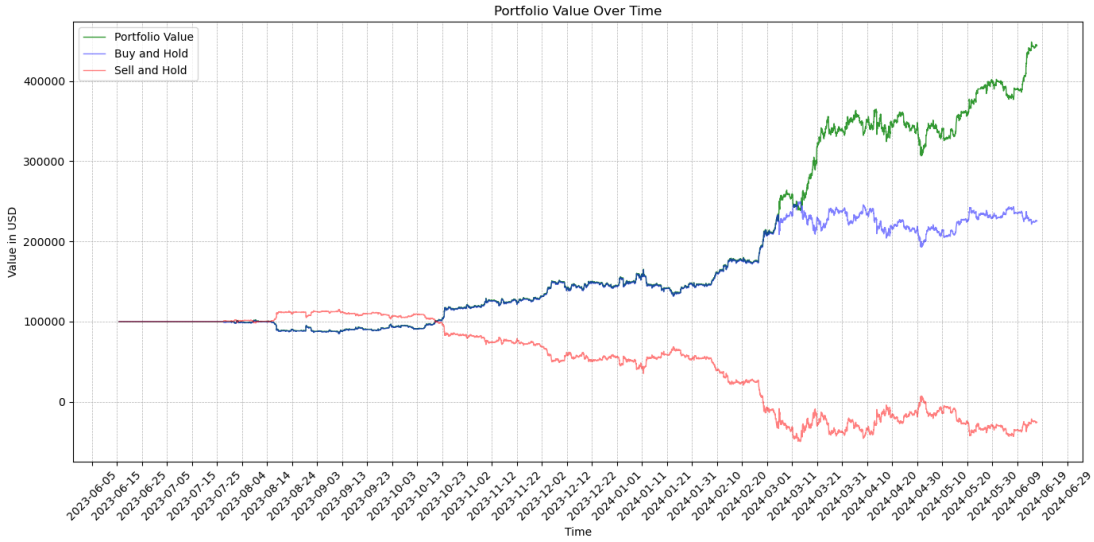
```
Number of Trades : 1

Portfolio value: 225529.8233089719


Results for Buy and Hold Strategy:
Total Return: 125.98%
Annualized Return: 149.57%
Annualized Sharpe Ratio: 3.14
Sortino Ratio: 3.99
Max Drawdown: 22.79%
Calmar Ratio: 6.56
```

(ב) ביצועי Buy and Hold

לאחר מכן התוכנית מציגה גרף של השינוי בערך התיק כפונקציה של זמן לכל שלושת האסטרטגיות כדי שנוכל להשוות בין האסטרטגיות. כפי שאפשר לראות בעמוד הבא.



איור 8: ערך התיק כפונקציה של זמן

7 ביבליוגרפיה וקישורים

- Chronos - Amazon paper
- Binance
- Our GitHub Repository