Qlib : An AI-oriented Quantitative Investment Platform

Xiao Yang , **Weiqing Liu** , **Dong Zhou** , **Jiang Bian** and **Tie-Yan Liu**
Microsoft Research
{Xiao.Yang, Weiqing.Liu, Zhou.Dong, Jiang.Bian, Tie-Yan.Liu}@microsoft.com

Abstract

Quantitative investment aims to maximize the return and minimize the risk in a sequential trading period over a set of financial instruments. Recently, inspired by rapid development and great potential of AI technologies in generating remarkable innovation in quantitative investment, there has been increasing adoption of AI-driven workflow for quantitative research and practical investment. In the meantime of enriching the quantitative investment methodology, AI technologies have raised new challenges to the quantitative investment system. Particularly, the new learning paradigms for quantitative investment call for an infrastructure upgrade to accommodate the renovated workflow; moreover, the data-driven nature of AI technologies indeed indicates a requirement of the infrastructure with more powerful performance; additionally, there exist some unique challenges for applying AI technologies to solve different tasks in the financial scenarios. To address these challenges and bridge the gap between AI technologies and quantitative investment, we design and develop Qlib that aims to realize the potential, empower the research, and create the value of AI technologies in quantitative investment.

1 Introduction

Quantitative investment, one of the hottest research fields, has been attracting numerous brilliant minds from both the academia and financial industry. In the last decades, with continuous efforts in optimizing the quantitative methodology, the whole community of professional investors has summarized a well-established yet imperfect quantitative research workflow. Recently, emerging AI technologies start a new trend in this research field. With increasing attention to exploring AI's great potential in quantitative investment, AI technologies have been widely adopted in the practical investment by quantitative researchers.

While AI technologies have been enriching the quantitative investment methodology, they also put forward new challenges to the quantitative investment system from multiple perspectives. First, the technological revolution in the quantitative investment workflow, caused by the flexibility of AI technologies, tends to require new supportive infrastructure. For example, while the traditional quantitative investment usually splits the whole workflow into a couple of sub-tasks, including stock trend prediction, portfolio optimization, etc., AI technologies make it possible to establish an end-to-end solution that generates the final portfolio directly. To support such end-to-end solution, it is necessary to upgrade the current infrastructure due to its data-driven nature.

Meanwhile, the AI technologies have to deal with the unique problems in some new scenarios, which require both plenty of domain knowledge in finance and rich experience in data science. Applying the solutions to quantitative research tasks without any domain adaptation rarely works. Such a circumstance leads to urgent demands for a platform to accommodate such a modern quantitative research workflow in the age of AI and provide guidance for the application of AI technologies in the financial scenario.

Therefore, we propose a new AI-oriented Quantitative Investment Platform called Qlib[1]. It aims to assist the research efforts of exploring the great potential of AI technologies in quantitative investment as well as empower quantitative researchers to create more significant values on AI-driven quantitative investment. Specifically, the AI-oriented framework of Qlib is designed to accommodate the AI-based solutions. Moreover, it provides high-performance infrastructure dedicated for quantitative investment scenario, which makes many AI research topics possible. In addition, a batch of tools designed for machine learning in the quantitative investment scenario is integrated with Qlib to benefit users in making fully utilization of AI technologies.

At last, we demonstrate some use cases and evaluate the performance of the infrastructure of Qlib by comparing several solutions for a typical task in quantitative investment. The results show the infrastructure of Qlib dedicated to quantitative investment outperforms most of existing solutions on this task.

2 Background and Related Works

In this section, we will first demonstrate the major practical problems of a modern quantitative researcher when applying

[1]The code is available at https://github.com/microsoft/qlib

of AI technologies in quantitative investment, which motivates the birth of Qlib. After that, we will briefly introduce the related work.

2.1 Practical Problems

Quantitative research workflow revolution

In the traditional investment research workflow, researchers often develop trading signals by linear models [Petkova, 2006] or manually designed rules[Murphy, 1999] based on several factors(factors are similar to features in machine learning) and basic financial data. And then, a trading strategy(typically Barra[Sheikh, 1996]) is followed to generate the target portfolio. At last, researchers evaluate the trading signal and portfolio by a back-testing function.

With the rise of AI technologies, it launches a technological revolution of traditional quantitative investment. The traditional quantitative research workflow is too primitive to accommodate such flexible technologies. In order to show the difference more intuitively, we'll demonstrate a typical modern research workflow based on AI technologies. It starts with a dataset with lots of features(typically more than hundreds of dimensions). Manually designing such amount of features takes lots of time. It is common to leverage machine learning algorithms to generate such features automatically[Potvin *et al.*, 2004; Neely *et al.*, 1997; Allen and Karjalainen, 1999; Kakushadze, 2016]. Generating data [Feng *et al.*, 2019] is another option for constructing a dataset. Based on diverse datasets, researchers have provided hundreds of machine learning methods to mine trading signals [Sezer *et al.*, 2019]. Researchers could generate the target portfolio based on such trading signals. But such a workflow is not the only choice. Instead of dividing a task into several stages, RL(reinforcement learning) provides an end-to-end solution from the data to the final trading actions directly [Deng *et al.*, 2016]. RL optimizes the trading strategy by interacting with the environment, which is a trading simulator in the financial scenario. RL needs a responsive simulator instead of a back-testing function in the traditional research workflow. Moreover, most of the AI algorithms have complicated hyperparameters, which need to be tuned carefully.

AI technologies are so flexible and already beyond the scope of existing tools designed for traditional methodologies. Building a research workflow based on AI technologies from scratch takes much time.

High performance requirements for infrastructure

With the emerging of AI technologies, the requirements for infrastructure have changed. Such a data-driven method could leverage a huge amount of data. The amount of data could reach the order of TB magnitude in the scenario of high-frequency trading. Besides, it is very common to derive thousands of new features (e.g., Alpha101 [Kakushadze, 2016]) from the basic price and volume data, which consist of only five dimensions in total. Some researchers even try to create new factors or features by searching expressions [Allen and Karjalainen, 1999; Neely *et al.*, 1997; Potvin *et al.*, 2004]. Such heavy work of data processing overburdens the researchers and even make some research topics impossible. Such circumstances put forward more stringent performance requirements for the infrastructure.

Obstacles to apply machine learning solutions

The financial data and task have their uniqueness and challenges. Applying the machine learning solutions to quantitative research tasks without any adaptation rarely works. Due to the extremely low SNR(Signal to Noise Ratio) in financial data, it is very hard to build a successful data-driven strategy in financial markets. Most machine learning algorithms are data-driven and have to deal with such difficulties. Without carefully handling the details, machine learning models can hardly achieve satisfying performance. Even a minor mistake can make the model over-fit the noise rather than learn effective patterns. Rightly handling the details requires a lot of domain knowledge of the financial industry. Moreover, the typical objectives, such as annualized return, are often not differentiable, which makes it hard to train models directly for machine learning methods. Defining a reasonable task with appropriate supervised targets is very important for modeling the finance data. Such barriers daunt quite a lot of data scientists without much domain knowledge of the financial industry.

Another necessary step to build a machine learning application is hyperparameter optimization. Different machine learning algorithms have different hyperparameter search spaces, each of which has multiple dimensions with different meanings and priorities. Some of the quantitative researchers come from the traditional financial industry and don't have much knowledge about machine learning. Such huge learning cost stops many users from giving full play to the maximum value of machine learning.

2.2 Related Work

In the financial industry, an investment strategy will become less profitable with more investors following it. Therefore, the financial practitioners, especially quantitative researchers, are never keen to share their own algorithms and tools. OLPS [Li *et al.*, 2016] is the first open-source toolbox for portfolio selection. It consists of a family of classical strategies powered by machine learning algorithms as benchmarks and toolkit to facilitate the development of new learning methods. This toolbox only supports Matlab and Octave, which is not compatible with current scientific mainstream language Python and thus not friendly to the modern machine learning algorithms. Its framework is quite simple, and modern quantitative research workflow based on AI technologies is much more complicated. Other quantitative tools emerge in recent years. QuantLib [Firth, 2004] only focuses part of modern quantitative research workflow. QUANTAXIS [2] focuses more on the IT infrastructure instead of the research workflow. Quantopian releases a series of open-source tools 1) Alphalens: a Python Library for performance analysis of predictive (alpha) stock factors 2) Zipline: an event-driven system for back-testing 3) Pyfolio: a Python library for performance and risk analysis of financial portfolios. All of them only focus on the analysis of trading signals or an investment portfolio.

[2] https://github.com/QUANTAXIS/QUANTAXIS

Overall, Qlib is the first open-source platform that accommodates the workflow of a modern quantitative researcher in the age of AI. It aims to empower every quantitative researcher to realize the great potential of AI technologies in quantitative investment.

3 AI-oriented Quantitative Investment Platform

3.1 Overall Design

In the cooperation with the quantitative researcher with years of hands-on experience in the financial market, we've encountered all of the above problems and explored all kinds of solutions. Motivated by current circumstances, we implement Qlib to apply AI technologies in quantitative investment.

AI-oriented framework Qlib is designed in a modularized way based on modern research workflow to provide the maximum flexibility to accommodate AI technologies. Quantitative researchers could extend the modules and build a workflow to try their ideas efficiently. In each module, Qlib provides several default implementation choices which work very well in practical investment. With these off-the-shelf modules, quantitative researchers could focus on the problem they are interested in a specific module without distracted by other trivial details. Besides code, computation and data can also be shared in some modules, so Qlib is designed to serve users as a platform rather than a toolbox.

High-performance infrastructure The performance of data processing is important to data-driven methods like AI technologies. As an AI-oriented platform, Qlib provides a high-performance data infrastructure. Qlib provides a time-series flat-file database [3]. Such a database is dedicated to scientific computing on finance data. It greatly outperforms current popular storage solutions like general-purpose databases and time-series databases on some typical data processing tasks in quantitative investment research. Furthermore, the database provides an expression engine, which could accelerate the implementation and computation of factors/features, which make research topics that rely on expressions computation possible.

Guidance for machine learning Qlib has been integrated with some typical datasets for quantitative investment, on which typical machine learning algorithms could successfully learn patterns with generalization ability. Qlib provides some basic guidance for machine learning users and integrates some reasonable tasks which consist of reasonable feature space and target label. Some typical hyperparameter optimization tools are provided. With guidance and reasonable settings, machine learning models could learn patterns with better generalization ability instead of just over-fitting the noise.

3.2 AI-oriented Framework

Figure 1 shows the overall framework of Qlib. This framework aims to 1) accommodate the modern AI technology, 2) help the quantitative researchers build a whole research workflow with minimal efforts 3) and leave them the maximal flexibility to explore problems they are interested without getting distracted by other parts.

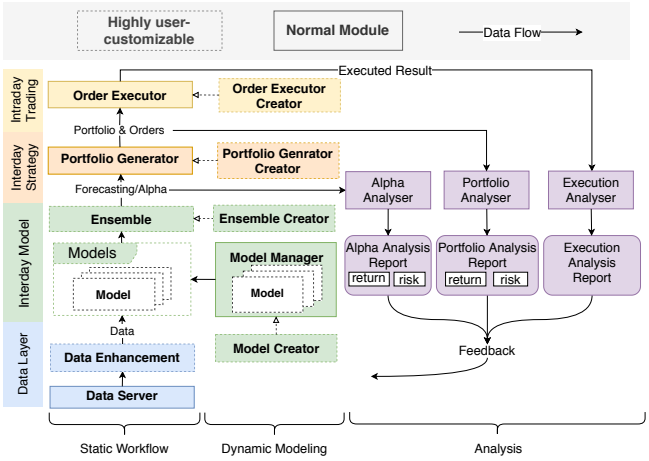

Figure 1: modules and a typical workflow built with Qlib

Such a target leads to a modularized design from the perspective of system design. The system is split into several individual modules based on the modern practical research workflow. Most of the quantitative investment research directions, no matter traditional or AI-based, could be regarded as implementations of one or multiple modules' interfaces. Qlib provides several typical implementations that work well in practical investment for users in each module . Moreover, the modules provide the flexibility for researchers to override existing methods to explore new ideas. With such a framework, researchers could try new ideas and test the overall performance with other modules with minimal cost.

The modules of Qlib are listed in Figure 1 and connected in a typical workflow. Each module corresponds to a typical sub-task in quantitative investment. A implementation in the module can be regarded as a solution for this task. We'll introduce each module and give some related examples of existing quantitative research to show how Qlib accommodate them.

It starts with the **Data Server module** in the bottom left corner, which provides a data engine to query and process raw data. With retrieved data, researcher could build his own dataset in the **Data Enhancement module**. Researchers have tried a lot solutions to build better datasets by exploring and constructing effective factors/features[Potvin *et al.*, 2004; Neely *et al.*, 1997; Allen and Karjalainen, 1999; Kakushadze, 2016]. Generating datasets for training[Feng *et al.*, 2019] is another research direction to provide datasets solution. The **Model Creater module** learns models based on datasets. In recent years, numerous researchers have explored all kinds of models to mine trading signals from financial dataset[Sezer *et al.*, 2019]. Moreover, meta-learning [Vilalta and Drissi, 2002] that tries to learn to learn provides a new learning paradigm for the Model Creator module. Given plenty of methods to model the financial data in a modern research workflow, the model management system has become a nec-

[3] https://en.wikipedia.org/wiki/Flat-file_database

essary part of the workflow. The **Model Manager module** is designed to handle such problems for modern quantitative researchers. With diverse models, ensemble learning is quite an effective way to enhance the performance and robustness of machine learning models, and it is frequently used in the financial area[Qiu *et al.*, 2014; Yang *et al.*, 2017; Zhao *et al.*, 2017]. It is supported by **Model Ensemble module**. **Portfolio Generator module** aims to generate a portfolio from trading signals output by models, which is known as portfolio management[Qian *et al.*, 2007]. Barra [Sheikh, 1996] provides the most popular solution for this task. With the target portfolio, we provide a high-fidelity trading simulator, **Orders Executor module**, to examine the performance of a strategy and **Analyser modules** to automatically analyze the trading signals, portfolio and execution results. The Order Executor module is designed as a responsive simulator rather than a back-testing function, which could provide the infrastructure for some learning paradigm(e.g., RL) that requires feedback of the environment produced by the Analyser modules.

The data in quantitative investment are in time-series format and updated by time. The size of in-sample dataset increases by time. A typical practice to leverage the new data is to update our models regularly [Wang *et al.*, 2019b]. Besides better utilization of increasing in-sample data, dynamically updating models [Yang *et al.*, 2019] and trading strategies [Wang *et al.*, 2019a] will improve the performance further due to dynamic nature of the stock market[Adam *et al.*, 2016]. Therefore, it is obviously not the optimal solution to use a set of static model and trading strategies in **Static Workflow**. Dynamic updating of models and strategies is a important research direction in quantitative investment. The modules in the **Dynamic Modeling** provide interfaces and infrastructure to accommodate such solutions.

3.3 High Performance Infrastructure

Financial data

We'll summarise the data requirements in quantitative research in this section. In quantitative research, the most frequently-used format of data follow such format

$$BasicData_T = \{x_{i,t,a}\}, i \in Inst, t \in Time, a \in Attr$$

where $x_{i,t,a}$ is the value of basic type(e.g. float, int), $Inst$ denotes the financial instruments set(e.g. stock, option, etc.), $Time$ denotes the timestampes set(e.g. trading days of stock market), $Attr$ denotes the possible attributes set of an instrument(e.g. open price, volume, market value), T denote the latest timestamp of the data(e.g. the latest trading date). $x_{i,t,a}$ denotes the value of attribute a of instrument i at time t.

Besides, instruments pools are necessary information to specify a set of financial instruments which change over time

$$Pool_T = \{pool_t\}, t \in Time, pool_t \subseteq Inst$$

S&P 500 Index [4] is a typical example of $Pool$.

Data update is an essential feature. The existing historical data will not change over time. Only the append operation of new data is necessary. The formalized update operation is

$$\begin{aligned} BasicData_T &= OldBasicData_T \cup \{x_{i,t,a_{new}}\} \\ BasicData_{T+1} &= BasicData_T \cup \{x_{i,T+1,a}\} \\ Pool_{T+1} &= Pool_T \cup \{pool_{t+1}\} \end{aligned}$$

User queries can be formalized as

$$\begin{aligned} Data_{Query} = \{x_{i,t,a} | i_t \in pool_t, pool_t \in Pool_{query} \\ a \in Attr_{query}, time_{start} \leq t \leq time_{end}\} \end{aligned}$$

which represents data query of some attributes of instruments in a specific time range in a specific pool.

Such requirements are quite simple. Many off-the-shelf open-source solutions support such operations. We classify them into three categories and list the popular implementations in each category.

- General-purpose database: MySQL[MySQL, 2001], MongoDB[Chodorow, 2013]
- Time-series database: InfluxDB [Naqvi *et al.*, 2017]
- Data file for scientific computing: Data organized by numpy[Oliphant, 2006] array or pandas[McKinney, 2011] dataframe

The general-purpose database supports data with diverse formats and structures. Besides, it provides lots of sophisticated mechanisms, such as indexing, transaction, entity-relationship model, etc. Most of them add heavy dependencies and unnecessary complexity to a specific task rather than solving the key problems in a specific scenario. The time-series database optimizes the data structures and queries for time-series data. But they are still not designed for quantitative research, where the data are usually in compact array-based format for scientific computation to take advantage of hardware acceleration. It will save a great amount of time if the data keep the compact array-based format from the disk to the end of clients without format transformation. However, both general-purpose and time-series database store and transfer the data in a different format for the general purpose, which is inefficient for scientific computation.

Due to the inefficiency of databases, array-based data gain popularity in the scientific community. Numpy array and pandas dataframe are the mainstream implementations in scientific computation, which are often stored as HDF[5] or pickle[6] on the disk. Data in such formats have light dependencies and are very efficient for scientific computing. However, such data are stored in a single file and hard to update or query.

After an investigation of above storage solutions, we find none could fit the quantitative research scenario very well. It is necessary to design a customized solution for quantitative research.

File storage design

Figure 2 demonstrates the file storage design. As shown in the left part of the figure, Qlib organize files in a tree structure. Data are separated into folders and files according to different

[4] https://en.wikipedia.org/wiki/S%26P_500_Index

[5] https://en.wikipedia.org/wiki/Hierarchical_Data_Format

[6] https://docs.python.org/3/library/pickle.html

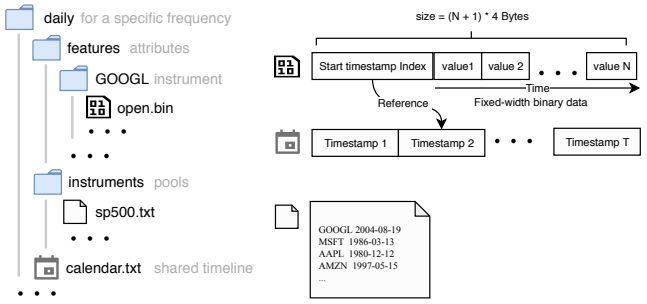

Figure 2: The description of the flat-file database; the left part is the structure of files; the right part is the content of files

frequencies, instruments and attributes. All the values of attributes are stored in binary data in a compact fixed-width format so that indexing by bytes becomes possible. The shared timeline is stored separately in a file named "calendar.txt". The data file of attribute values sets its first 4 bytes to the index value of the timeline to indicates the start timestamp of the series of data. With the start time index, Qlib could align all the values on the time dimension.

The data are stored in a compact format, which is efficient to be combined into arrays for scientific computation. While it achieves high performance like array-based data in scientific computation, it meets data update requirements in the quantitative investment scenario. All data are arranged in the order of time. New data could be updated by appending, which is quite efficient. Adding and removing attributes or instruments are quite straightforward and efficient, because they are stored in separate files. Such a design is extremely light-weighted. Without the overheads of databases, Qlib achieves high performance.

Expression Engine

It is quite a common task to develop new factors/features based on basic data. Such a task takes a large proportion of the time of many quantitative researchers. Both Implement such factors by code, and the computation process is time-consuming. Therefore, Qlib provides an expression engine to minimize the effort of such tasks.

Actually, the nature of factors/features is a function that transforms the basic data into the target values. The function could break down into a combination of a series of expressions. The expression engine is designed based on this idea. With this expression engine, quantitative researchers could implement new factors/features by writing expressions instead of complicated code. For example, The Bollinger Band technical indicator [Bollinger, 2002] is a widely used technical factor and its upper bounds can be implemented by just a simple expression *"(MEAN($close, N)+2*STD($close, N)-$close)/MEAN($close, N)"* with the expression engine.

Such an implementation is simple, readable, reusable and maintainable. Users can easily build a dataset with just a series of simple expressions. Searching expressions to construct effective trading signals is a typical research topic, which has been explored by many researchers [Allen and Karjalainen, 1999; Neely *et al.*, 1997; Potvin *et al.*, 2004]. An expression engine is an essential tool for such a research topic.

Cache system

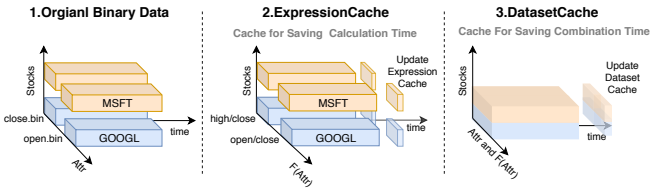

Figure 3: The disk cache system of Qlib; expression cache for saving time of expression computation; dataset cache for saving time of data combination

To avoid replicated computation, Qlib has a built-in cache system. It consists of memory cache and disk cache.

In-memory cache When Qlib computes factors/features with its expression engine, it parses the expression into a syntax tree. All computed results of nodes will be stored in an LRU(Least Recently Used) cache in memory. The replicated computation of same (sub-)expressions can be saved.

Disk cache A typical workflow of data processing in quantitative investment can be divided into three steps: fetching original data, computing expressions and combining data into arrays for scientific computation. Computing expressions and combining data are very time-consuming. It could save much time if we can cache the shared intermediate data. In practical data processing tasks, many intermediate results can be shared. For example, the same expression computation can be shared by different data processing tasks. Therefore Qlib designed a 2-level disk cache mechanism. The cache system is shown in Figure 3. The left part is the original data we described in Section 3.3. The first level is expression cache, which will save all the computed expressions to the disk cache. The data structure of the expression cache is the same as the original data. With the expression cache, the same expression will be computed only once. After the expression cache is dataset cache, which stores the combined data to save the combination time. The cache data of both levels are arranged by time and indexable on the time dimension, so the disk cache can be shared even when the query time changes. Moreover, Qlib support data update by appending new data thanks to the data arrangement by time. The maintenance of the data is much easier with such a mechanism.

3.4 Guidance for Machine Learning

As we discussed in Section 2, guidance for machine learning algorithms is very important. Qlib provides typical datasets for machine learning algorithms. Some typical task settings can be found in Qlib , such as data pre-processing, learning targets, etc. Researchers don't have to explore everything from scratch. Such guidances provide lots of domain knowledge for researchers to start their journey in this research area.

For most machine learning algorithms, hyperparameter optimization is a necessary step to achieve better generalization. Although it is important, it takes a lot of effort and is

	HDF5	MySQL	MongoDB	InfluxDB	Qlib -E -D	Qlib +E -D	Qlib +E +D
Storage(MB)	**287**	1,332	911	**394**	**303**	802	1,000
Load Data(s)	**0.80±0.22**	182.5±4.2	70.3±4.9	186.5±1.5	**0.95±0.05**	4.9±0.07	7.4±0.3
Compute Expr.(s)	179.8±4.4				137.7±7.6	**35.3±2.3**	-
Convert Index(s)	-				3.6±0.1		-
Filter by Pool(s)	3.39 ±0.24						-
Combine data(s)	1.19±0.30						-
Total (1CPU) (s)	184.4±3.7	365.3±7.5	253.6±6.7	368.2±3.6	147.0±8.8	47.6±1.0	**7.4±0.3**
Total(64CPUs) (s)	-				8.8±0.6	**4.2±0.2**	-

Table 1: Performance comparison of different storage solutions

quite repetitive. Therefore, Qlib provides a **Hyperparameters Tuning Engine(HTE)** to make such a task easier. HTE provides an interface to define a hyperparameter search space Θ and then search the best hyperparameters θ automatically.

In a typical financial task of modeling time-series data, the new data comes in sequence by time. To leverage the new data, models have to be re-trained on new data periodically. The new best hyperparameters θ change but are often close to previous best hyperparameters. HTE provides a mechanism dedicated to hyperparameter optimization on financial tasks. It generates a new distribution for hyperparameters search space for better a chance to reach the best point with fewer trials. The distribution for searching θ can be formalized as

$$p_{new}(x) = \frac{p_{prior}(x)\varphi_{\theta_{prev},\sigma^2}(x)}{\mathbb{E}_{x\sim p_{prior}}[\varphi_{\theta_{prev},\sigma^2}(x)]}$$

where p_{prior} is the original hyperparameters search space; $\varphi_{\theta_{prev},\sigma^2}(x) \sim \mathcal{N}(\theta_{prev}, \sigma^2)$; θ_{prev} is the best hyperparameter in last model training. The domain of hyperparameter search space remains the same, but the probability density around θ_{prev} increases.

4 Use Case & Performance Evaluation

4.1 Use Case

Qlib provide a **Config-Driven Pipeline Engine(CDPE)** to help researchers build the whole research workflow show in Figure 1 easier. The user could define a workflow with just a simple config file like List **??**(some trivial details are replaced by "..."). Such an interface is not mandatory, and we leave the maximal flexibility to users to build a quantitative research workflow by code like building blocks.

4.2 Performance Evaluation

The performance of data processing is important to data-driven methods like AI technologies. As an AI-oriented platform, Qlib provides a solution for data storage and data processing. To demonstrate the performance of Qlib, We compare Qlib with several other solutions discussed in Section 3.3, which includes *HDF5, MySQL, MongoDB, InfluxDb and Qlib*. The *Qlib +E -D* indicates Qlib with expression cache enabled and dataset cache disabled, and so forth.

```
model:
  class:    "qlib.model.GBDTModel"
  args:     ...
data:
  class:    "qlib.dataset.Alpha360"
learner:
  class:    "qlib.trainer.NormalTrainer"
  args:     ...
portfolio_manager:
  class:    "qlib.portfolio.TopkStrategy"
executor:
  account: ...
```

Figure 4: A Configuration example of CDPE

The task for the solutions is to create a dataset from the basic OHLCV[7] daily data of a stock market, which involves data query and processing. The final dataset consists of 14 factors/features derived from OHLCV data(e.g. "Std($close, 5)/$close"). The time of the data ranges from 1/1/2007 to 1/1/2020. The stock pool consists of 800 stocks each day, which changes daily.

Besides the comparison of the total time of each solution, we break down the task into following steps for more details.

- **Load Data** Load the OHCLV data or cache into RAM as the array-based format for scientific computation.
- **Compute Expr.** Compute the derived factors/features.
- **Convert Index** It only applies to Qlib. Because Qlib doesn't store the indices(i.e., timestamp, stock id) in the original data, it has to set up data indices.
- **Filter data** Filter the stock data by a specific pool. For example, SP500 involves more than 1 thousand stock in total, but it only includes 500 stocks daily. The data not included in SP500 on a specific day should be filtered out, though it has ever been in SP500. It is impossible to filter out data when loading data, because some derived features rely on historical OHLCV data.
- **Combine data** Concatenate all the data of different stocks into a single piece of array-based data

As we can seen in Table 1. Qlib's compact storage achieves similar size and loading speed as the dedicated scientific

[7]The open, high, low, close price and trading volume of a stock

HDF5 data file. The databases take too much time on loading data. After looking into the underlying implementation, we find that data go through too many layers of interfaces and unnecessary format transformations in both general-purpose database and time-series database solution. Such overheads greatly slow down the data loading process. Due to the memory cache of Qlib, Qlib -E -D saves about 24% of the time of Compute Expr. Moreover, Qlib provides expression cache and dataset cache mechanism. With expression cache enabled in Qlib +E -D, 80.4% of the time for Compute Expr. is saved if no expression cache is missed. Combining the factors/features into one piece of array-based data for each stock accounts for the major time consuming of Qlib +E -D, which is included in the Compute Expr. step. Besides the computation cost, the most time-consuming step is data combination. The dataset cache is designed to reduce such overheads. As shown in the column Qlib +E +D, the time cost is further reduced.

Moreover, Qlib can leverage multiple CPU cores to accelerate computation. As we can see in the last line of Tabel 1, the time cost is significantly reduced for Qlib with multiple CPUs. Qlib +E +D can't be accelerated further due to it just reads the existing cache and almost computes nothing.

4.3 More about Qlib

Qlib an opensource platform in continuous development. More detailed documentations can be found in its github repository [8]. A lot of features(e.g. data service with client-server architecture, analysis system, automatic deployment on the cloud) not introduced in detail in this paper could be found in the online repository. Your contributions are welcomed.

5 Conclusion

In this paper, we present practical problems of modern quantitative researchers in the age of AI. Based on these practical problems, we design and implement Qlib that aims to empower every quantitative researcher to realize the great potential of AI-technologies in quantitative investment.

References

[Adam *et al.*, 2016] Klaus Adam, Albert Marcet, and Juan Pablo Nicolini. *Stock market volatility and learning*, 2016.

[Allen and Karjalainen, 1999] Franklin Allen and Risto Karjalainen. Using genetic algorithms to find technical trading rules. *Journal of financial Economics*, 51(2):245–271, 1999.

[Bollinger, 2002] John Bollinger. *Bollinger on Bollinger bands*. McGraw Hill Professional, 2002.

[Chodorow, 2013] Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage*. " O'Reilly Media, Inc.", 2013.

[Deng *et al.*, 2016] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.

[Feng *et al.*, 2019] Fuli Feng, Huimin Chen, Xiangnan He, Ji Ding, Maosong Sun, and Tat-Seng Chua. Enhancing stock movement prediction with adversarial training. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5843–5849. AAAI Press, 2019.

[Firth, 2004] N Firth. Why use quantlib. *Paper available at: http://www. quantlib. co. uk/publications/quantlib. pdf*, 2004.

[Kakushadze, 2016] Zura Kakushadze. 101 formulaic alphas. *Wilmott*, 2016(84):72–81, 2016.

[Li *et al.*, 2016] Bin Li, Doyen Sahoo, and Steven CH Hoi. Olps: a toolbox for on-line portfolio selection. *The Journal of Machine Learning Research*, 17(1):1242–1246, 2016.

[McKinney, 2011] Wes McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14, 2011.

[Murphy, 1999] John J Murphy. *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin, 1999.

[MySQL, 2001] AB MySQL. Mysql, 2001.

[Naqvi *et al.*, 2017] Syeda Noor Zehra Naqvi, Sofia Yfantidou, and Esteban Zimányi. Time series databases and influxdb. *Studienarbeit, Université Libre de Bruxelles*, 2017.

[Neely *et al.*, 1997] Christopher Neely, Paul Weller, and Rob Dittmar. Is technical analysis in the foreign exchange market profitable? a genetic programming approach. *Journal of financial and Quantitative Analysis*, 32(4):405–426, 1997.

[Oliphant, 2006] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[Petkova, 2006] Ralitsa Petkova. Do the fama–french factors proxy for innovations in predictive variables? *The Journal of Finance*, 61(2):581–612, 2006.

[Potvin *et al.*, 2004] Jean-Yves Potvin, Patrick Soriano, and Maxime Vallée. Generating trading rules on the stock markets with genetic programming. *Computers & Operations Research*, 31(7):1033–1047, 2004.

[Qian *et al.*, 2007] Edward E Qian, Ronald H Hua, and Eric H Sorensen. *Quantitative equity portfolio management: modern techniques and applications*. CRC Press, 2007.

[Qiu *et al.*, 2014] Xueheng Qiu, Le Zhang, Ye Ren, Ponnuthurai N Suganthan, and Gehan Amaratunga. Ensemble deep learning for regression and time series forecasting. In *2014 IEEE symposium on computational intelligence in ensemble learning (CIEL)*, pages 1–6. IEEE, 2014.

[8] https://github.com/microsoft/qlib/

[Sezer *et al.*, 2019] Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning: A systematic literature review: 2005-2019. *arXiv preprint arXiv:1911.13288*, 2019.

[Sheikh, 1996] Aamir Sheikh. Barra's risk models. *Barra Research Insights*, pages 1–24, 1996.

[Vilalta and Drissi, 2002] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

[Wang *et al.*, 2019a] Lewen Wang, Weiqing Liu, Xiao Yang, and Jiang Bian. Conservative or aggressive? confidence-aware dynamic portfolio construction. In *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1–5. IEEE, 2019.

[Wang *et al.*, 2019b] Shouxiang Wang, Xuan Wang, Shaomin Wang, and Dan Wang. Bi-directional long short-term memory method based on attention mechanism and rolling update for short-term load forecasting. *International Journal of Electrical Power & Energy Systems*, 109:470–479, 2019.

[Yang *et al.*, 2017] Bing Yang, Zi-Jia Gong, and Wenqi Yang. Stock market index prediction using deep neural network ensemble. In *2017 36th Chinese Control Conference (CCC)*, pages 3882–3887. IEEE, 2017.

[Yang *et al.*, 2019] Xiao Yang, Weiqing Liu, Lewen Wang, Cheng Qu, and Jiang Bian. A divide-and-conquer framework for attention-based combination of multiple investment strategies. In *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1–5. IEEE, 2019.

[Zhao *et al.*, 2017] Yang Zhao, Jianping Li, and Lean Yu. A deep learning ensemble approach for crude oil price forecasting. *Energy Economics*, 66:9–16, 2017.