The Machine Learning Model Development Lifecycle for Prediction of Electrical Energy Market Volumes

Anatoliy Doroshenko[1,2,*,†], Dmitry Zhora[1,†], and Oleksii Zhyrenkov[1,†]

[1] *Institute of Software Systems of the National Academy of Sciences of Ukraine, Glushkov Ave. 40, build. 5, Kyiv, 03187, Ukraine*

[2] *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Peremohy Ave. 37, Kyiv, 03056, Ukraine*

Abstract

The development of production-ready software solution that employs artificial intelligence is a complex incremental process that requires competence in multiple areas like business domain, programming, statistics, machine learning, containers, networking and deployment. This is a challenge for commercial companies as specialists with diverse qualifications and skills are required. This article highlights the modern state of electrical energy markets in Ukraine, and provides the comparative analysis of regression algorithms used for market volume forecasting. The development process is demonstrated from technical perspective: the dataset is analyzed and augmented with additional information, the optimal set of input parameters is determined, the best machine learning model is trained and serialized to file, the docker image is built with software layer that preloads the serialized model, the docker contained is deployed to Kubernetes cluster for real-time access via REST protocol.

Keywords

Machine learning, regression algorithms, model serialization, docker container, Kubernetes, MLOps, BentoML, Yatai, inference platform, model deployment, electrical energy markets.

1. Introduction

The usage of machine learning techniques [1] de-facto became a standard for modern systems that need to provide a forecast, classify data records or implement an associative search. In the long term, this approach is expected to provide significant economic benefits. One of the popular and established machine learning libraries is scikit-learn [2]. The regression algorithms available in this library are used in current research to build the forecasting model for electrical energy markets.

The increasing complexity of energy markets requires the advanced forecasting models to predict future trends accurately. These models are crucial for decision making, trading and planning in energy systems. However, the development of production-ready forecasting solution involves multiple stages -- each requiring the expertise in programming, mathematics, statistics, machine learning, containers, deployment systems and potentially in cloud technologies.

In this article we showcase the end-to-end machine learning workflow for forecasting the trade volume of electrical energy markets utilizing popular regression algorithm and modern deployment infrastructure. The workflow employs BentoML orchestration platform for model packaging and API hosting, Docker for containerization and Kubernetes for autoscaling. The aim is to provide a practical guide that can be applied in real-world environment, emphasizing ease of use, modularity, scalability, and interoperability.

Information Technology and Implementation (IT&I-2024), November 20-21, 2024, Kyiv, Ukraine

* Corresponding author.

† These authors contributed equally.

doroshenkoanatoliy2@gmail.com (A. Doroshenko); dmitry.zhora@gmx.com (D. Zhora); ozhyrenkov@gmail.com (O. Zhyrenkov)

0000-0002-8435-1451 (A. Doroshenko); 0009-0006-6073-7751 (D. Zhora); 0009-0007-3124-1359 (O. Zhyrenkov);

2. Electrical energy markets dataset

Historically, Ukraine was supporting only one market for electrical energy. This market of long-term bilateral agreements wasn't flexible enough to balance the interests of consumers and suppliers of electricity. On July 1st, 2019, Ukraine adopted the European model [3] that assumes the following four markets: bilateral, day-ahead, intraday, and balancing. Despite the electricity market models in Europe having some differences [4], this was a considerable progress in facilitating the electricity trading between countries.

	A	B	C	D	E	F
1	TradeDate	TradeHour	Bilateral	DayAhead	Intraday	Balancing
2	2020-07-01	0	11746.78	2494	1018.2	-2101
3	2020-07-01	1	11654.98	2697.1	660.3	-2249.5
4	2020-07-01	2	11606.28	2606.8	624	-2398.8
5	2020-07-01	3	11637.28	2507.7	614.7	-2681.5
6	2020-07-01	4	11614.58	2487.2	607.7	-2666.2
7	2020-07-01	5	11645.48	2629.3	605	-2832
8	2020-07-01	6	11696.58	2937	610.6	-2455.8
9	2020-07-01	7	12160.58	3110.5	690.3	-2275.5

Figure 1: The time-series data representing four electricity markets (megawatt-hours).

The bilateral market can be referenced also as a future or forward market. In Ukraine, as shown in Figure 1, the total amount of deals is recorded every hour. The markers are organized in a way to provide integrated access for all market participants and to balance energy price and volatility. For example, the bilateral market has lowest electricity price, high volume and low volatility. And vice-versa, the balancing market has highest average price, low volume and high volatility.

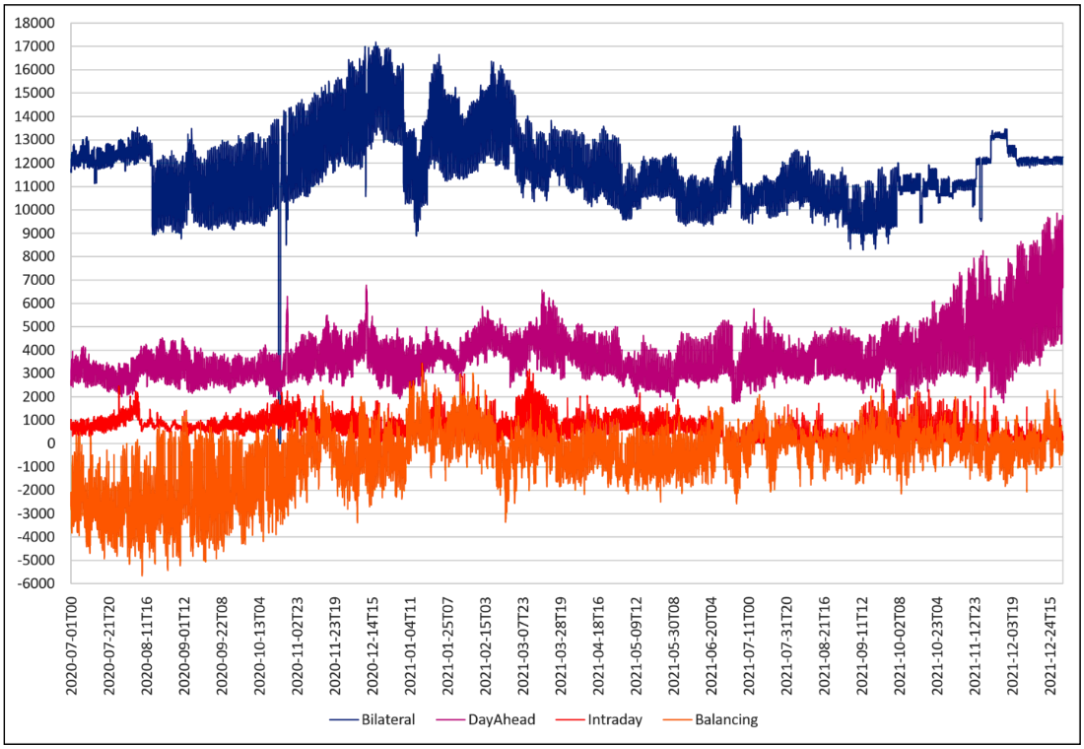

Figure 2: The dependency of market volume on time (megawatt-hours).

The large and complex electrical grids that belong to private or state enterprises still obey to the laws of physics. The amount of produced electrical energy is equal to amount of consumed energy, and this amount is exactly represented within the corresponding electricity market volume [5]. Also, if the amount of electrical energy traded and transmitted is measured on power substations then minor losses associated with electrical resistance can be disregarded. As a summary, for current application domain the following terms are equivalent: energy production, energy consumption and market volume.

The dataset used for this research matches the time range from July 1st, 2020, to December 31st, 2021. The corresponding market volume dynamics is shown above in figure 2. The volume of each of the four markets mentioned in the beginning of the section is calculated hourly in megawatts per hour. For comparison, some European markets record the trading data every 15 minutes.

3. Usage of additional parameters

It is common for real-world processes that the dynamics of monitored parameters is affected by other factors that are not available in the original dataset. In particular, the electricity production is influenced by outside temperature [6]. It was decided to add two temperature columns with hourly data representing the center of Ukraine and Kyiv, the corresponding chart is shown in figure 3.

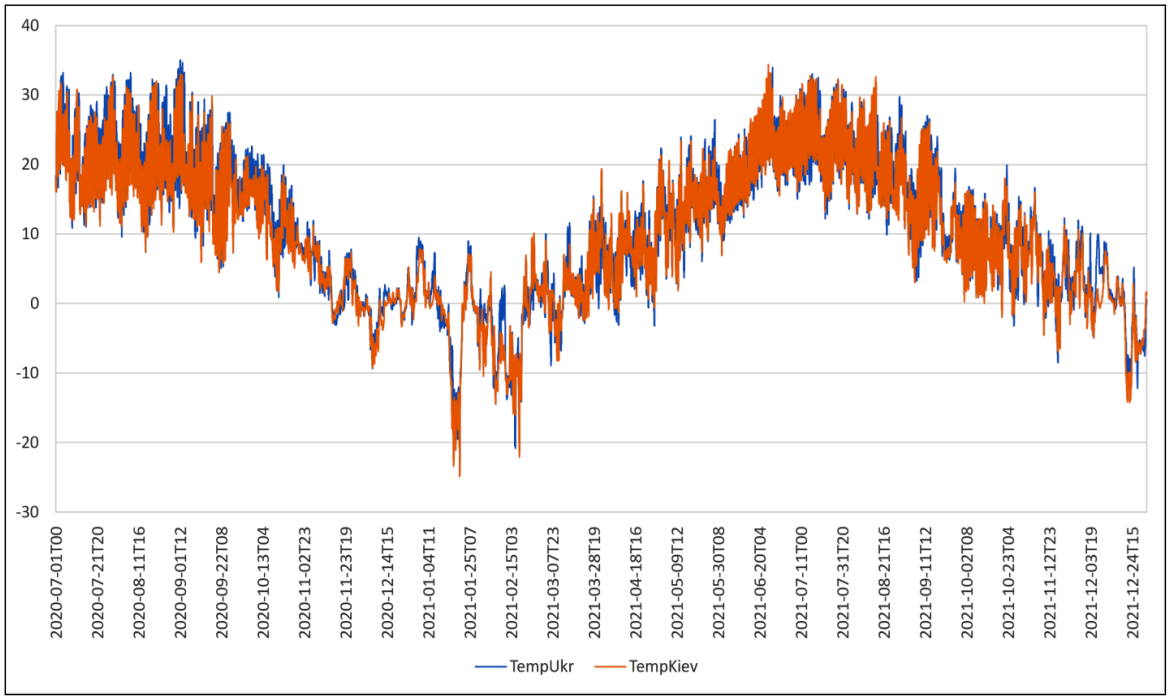

Figure 3: Dependency of outdoors temperature in Ukraine sampled hourly.

It is natural to assume periodic patterns in the consumption of electrical energy. Eventually, they represent the activity of final consumers. In particular, the following cycle types are possible: daily, weekly, monthly, and yearly. The challenge is to provide the representation of time in a way that close moments in time would be interpreted as close by machine learning algorithm.

The solution that is convenient from computational perspective [7] is to calculate sine and cosine functions when the corresponding argument represents the phase of the cycle. Obviously, the close values on the timescale are represented by close values of these periodic functions.

The augmented spreadsheet is shown in Figure 4. Besides the original four columns with market volume data ten other columns were added. These periodic data series were calculated with the help of an algorithm written in Python.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	TradeDate	TradeHour	Bilateral	DayAhead	Intraday	Balancing	TempUkr	TempKiev	SinDay	CosDay	SinWeek	CosWeek	SinMonth	CosMonth	SinYear	CosYear
2	2020-07-01	0	11746.78	2494	1018.2	-2101	18.38	17.99	0	1	0.9749	-0.2225	0	1	0.0172	-0.9999
3	2020-07-01	1	11654.98	2697.1	660.3	-2249.5	16.72	16.33	0.2588	0.9659	0.9659	-0.2588	0.0084	1	0.0165	-0.9999
4	2020-07-01	2	11606.28	2606.8	624	-2398.8	16.89	16.37	0.5	0.866	0.9556	-0.2948	0.0169	0.9999	0.0157	-0.9999
5	2020-07-01	3	11637.28	2507.7	614.7	-2681.5	16.99	16.52	0.7071	0.7071	0.9439	-0.3303	0.0253	0.9997	0.015	-0.9999
6	2020-07-01	4	11614.58	2487.2	607.7	-2666.2	16.94	16.49	0.866	0.5	0.9309	-0.3653	0.0338	0.9994	0.0143	-0.9999
7	2020-07-01	5	11645.48	2629.3	605	-2832	16.06	16.11	0.9659	0.2588	0.9166	-0.3999	0.0422	0.9991	0.0136	-0.9999
8	2020-07-01	6	11696.58	2937	610.6	-2455.8	16.06	16.3	1	0	0.901	-0.4339	0.0506	0.9987	0.0129	-0.9999
9	2020-07-01	7	12160.58	3110.5	690.3	-2275.5	16.12	16.26	0.9659	-0.2588	0.8841	-0.4673	0.0591	0.9983	0.0122	-0.9999

Figure 4: Augmented market volume dataset with temperature and periodic data.

4. Resampling of original data

As original data record contains the figures representing the current hour it makes sense to add two types of columns in the dataset: the parameters that represent the history and parameters that represent the future to be forecasted. It was a heuristic decision to consider up to 24 hours in both directions. The special naming convention was applied for new parameters. For example, the name BilateralM1 designates the bilateral market volume that was an hour ago in relation to current record under consideration. Similarly, the name BilateralP1 indicated the bilateral market volume in one hour. This additional information is expected improve the forecast accuracy.

The obtained dataset had 13'129 records. In particular, the first 24 data records and last 24 records were deleted as after resampling they did not contain all necessary parameters. The dataset was split into training and testing parts, the proportion of 80% to 20% was used in this case. The random split functionality is provided by scikit-learn library. The datasets were saved into files, so that different machine learning algorithms considered later are evaluated with equal conditions.

5. Model comparison metrics

Three metrics were used in this work to compare input parameter sets and different regression algorithms: R2 score (determination coefficient), MAPE (mean absolute percentage error) and MAE (mean absolute error). From the computational perspective each metric measures the discrepancy between test set and forecasted data for selected output column representing one of electrical energy trading volumes. The R2 score was used to make a decision, although these metrics were mainly correlated. The nearest neighbors regression algorithm was used to check the performance of input parameters, it provides quite competitive results and has limited number of hyperparameters to tune. Other algorithms available in scikit-learn library were evaluated as a next step.

The complexity of machine learning algorithm within this library is hidden behind fit and predict methods that have the same signature across many regression and classification algorithms. So, it is relatively simple to reuse these methods and to substitute one algorithm instead of another.

6. Feature selection approaches

In general, it is possible to select the set of input parameters manually. This workflow assumes adding or removing one parameter at a time and evaluating the performance of resulting model. This process is time consuming as it has 2^n combinations of parameters, here n represents the total number of possible inputs. The alternative is to use the automation facilities provided by machine learning library, in this case by scikit-learn. The following classes are worth mentioning: GridSearchCV, LassoCV and SelectFromModel. The latter option was used in current research.

The periodic parameters like SinDay were not added to the history as such parameters precisely indicate the moment in time, the history would provide just redundant information in this case. For every hour we have 4 parameters representing the electrical energy market volumes and 2 parameters representing the temperature. Thus, overall we have 6 * 24 = 144 input parameters to select from. The final and locally optimal set of input features obtained with SelectFromModel class

contained 60 entries out of 144 [6]. The R2 score was improved slightly and still was around 96% for the nearest neighbors algorithm. The positive outcome in this case is that model was somewhat simplified. The high dimensionality of input space is typically considered a problem. Thus, the removal of noisy parameters in majority of cases should be a positive step.

7. Prediction error distribution

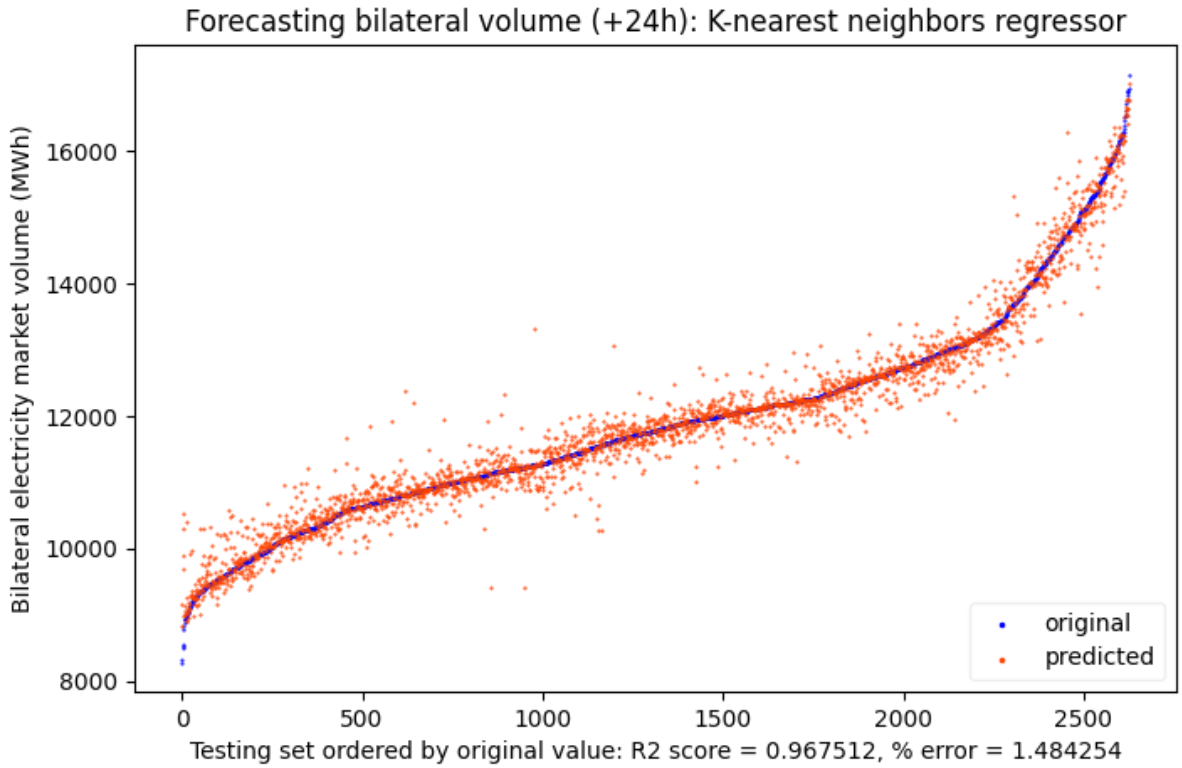

Figure 5: Forecasting error for one day ahead, bilateral market volume.

The bilateral market was selected as an example to demonstrate the error distribution. Its market volume prediction error on the test set is shown in figures 5 and 6. For convenience of representation the test set was sorted by original market volume. The predicted values are shown on the first chart with dots. The histogram allows to estimate the probability density of error distribution. It is unusual that obtained prediction error is not quite gaussian. In particular, this is the case for bilateral and intraday market volumes.

8. Selection of regression algorithm

Up to this point, only one algorithm was considered – the nearest neighbors regressor. Clearly, it makes sense to evaluate the performance of other algorithms on the same set of input parameters. The output parameters were selected for one day ahead forecasting: BilateralP24, DayAheadP24, IntradayP24 and BalancingP24. The prediction accuracy results are provided in tables 1-3 below. In particular, the comparison with the following established forecasting instruments is available: multi-layer perceptron **Error! Reference source not found.**, support v ector machine **Error! Reference source not found.**, and linear regression **Error! Reference source not found.**. It is worth noting that some algorithms do not natively support multi-output configuration, so it was needed to use the class MultiOutputRegressor to overcome this problem and cover four output parameters with one machine learning model.

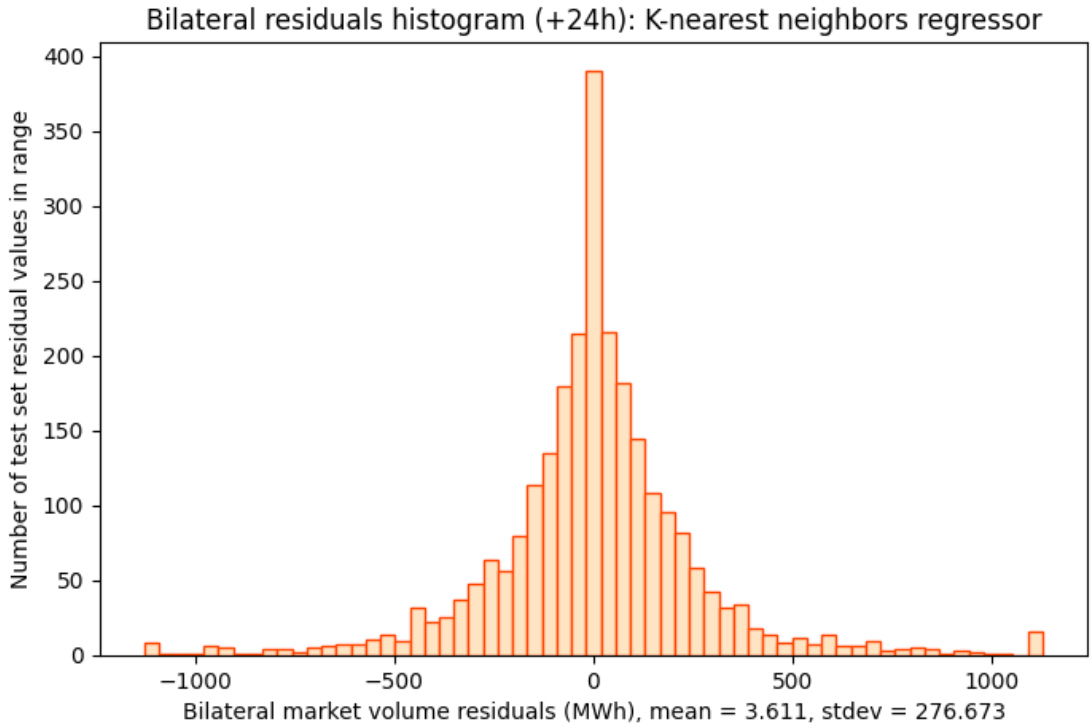

Figure 6: Residuals histogram for one day forecasting, bilateral market volume.

Table 1
R2 scores obtained for regression algorithms on the testing dataset

Regression Algorithm	Bilateral	DayAhead	Intraday	Balancing
Histogram Gradient Boosting	**0.987344**	**0.972738**	**0.878364**	**0.919632**
Ada Boost Regressor	0.980086	0.961343	0.851729	0.903254
Gradient Boosting Regressor	0.978789	0.963179	0.846663	0.901125
Extra Trees Regressor	0.974619	0.959632	0.864845	0.898156
Nearest Neighbors Regressor	0.967512	0.948956	0.860665	0.875551
Random Forest Regressor	0.966803	0.947184	0.831671	0.873048
Support Vector Machine	0.938416	0.907901	0.782819	0.785732
Multi-Layer Perceptron (QNO)	0.935896	0.904092	0.754444	0.791107
Multi-Layer Perceptron (SGD)	0.934140	0.908779	0.773580	0.815628
Elastic Net Regressor	0.929248	0.903003	0.755470	0.779082
Linear Regression	0.929214	0.902979	0.755526	0.779067
Bayes Ridge Regressor	0.925025	0.892584	0.741958	0.778845

For all three metrics considered in this work the winner algorithm is HistoramGradientBoosting regressor. It is one of the fastest methods as it is employing vector quantization technique to reduce the training set size. Another benefit is that it can natively process the datasets with missing values. The training phase for this algorithm and current dataset takes about 20 seconds, the inference or prediction phase takes less than a second. In general, the ensemble algorithms perform much better for this specific forecasting task.

Two types of multi-layer perceptron were tried on a dataset. Here QNO stands for quasi-Newton optimizer and SGD stands for stochastic gradient descend. In the first case the synaptic weights of neural network are determined as analytic solution to optimization task when the second-order approximation is calculated for the error function. In the second case the minimum (local or global) is determined with iterative descend process. It appears that for this task the 4-layer architecture performs better than 3 or 5-layer.

Table 2

Mean absolute percentage errors for selected regression algorithms

Regression Algorithm	Bilateral	DayAhead	Intraday	Balancing
Histogram Gradient Boosting	**0.009708**	**0.035550**	0.306800	**3.414739**
Ada Boost Regressor	0.010436	0.039889	**0.299648**	3.703527
Gradient Boosting Regressor	0.011671	0.041963	0.331089	4.306912
Extra Trees Regressor	0.013403	0.044706	0.397157	3.687793
Nearest Neighbors Regressor	0.014842	0.047414	0.312221	4.160123
Random Forest Regressor	0.015383	0.050163	0.444903	4.214244
Support Vector Machine	0.020497	0.065063	0.446288	5.010112
Multi-Layer Perceptron (QNO)	0.022011	0.068955	0.484654	4.251736
Multi-Layer Perceptron (SGD)	0.023281	0.067661	0.497681	4.585881
Elastic Net Regressor	0.021644	0.067856	0.460139	5.917427
Linear Regression	0.021679	0.067995	0.460715	5.929356
Bayes Ridge Regressor	0.022225	0.069814	0.501726	5.903614

It makes sense to explain the high mean absolute percentage error for the balancing column in table 2. This is not a mistake, these values appear be high because the balancing market volume often crosses zero line as presented in figure 2. The calculations according to MAPE formula above assume the division by these small values. It is better to interpret this situation that MAPE metric is simply not adequate with respect to balancing column.

Table 3
Mean absolute errors for selected algorithms in megawatt-hours

Regression Algorithm	Bilateral	DayAhead	Intraday	Balancing
Histogram Gradient Boosting	**114.528**	**136.017**	107.623	**287.495**
Ada Boost Regressor	122.856	151.671	**107.042**	308.036
Gradient Boosting Regressor	137.181	161.254	119.274	324.798
Extra Trees Regressor	156.724	165.609	118.335	320.904
Nearest Neighbors Regressor	175.124	183.691	112.622	344.727
Random Forest Regressor	180.816	187.724	131.586	360.659
Support Vector Machine	239.541	247.547	150.333	475.954
Multi-Layer Perceptron (QNO)	257.264	260.785	159.517	481.078
Multi-Layer Perceptron (SGD)	270.583	256.261	157.399	445.710
Elastic Net Regressor	253.327	259.857	155.518	487.916
Linear Regression	253.691	260.242	155.743	488.161
Bayes Ridge Regressor	260.956	269.083	160.748	487.498

9. Machine learning operations

The modern landscape of Machine Learning Operations (MLOps) emphasizes the integration of machine learning models into production environment, ensuring that they deliver consistent and reliable results. MLOps encompasses a set of practices that aim to automate and improve the deployment, monitoring, and management of ML models. The key principles include collaboration between data scientists and operations teams, continuous integration and deployment (CI/CD), and the use of standardized tools for model serving and monitoring.

Various ML inference tools have emerged to facilitate these processes, including TensorFlow Serving, MLflow, and BentoML [11]. Each tool offers unique features tailored to different aspects of the ML lifecycle. The table 4 highlights core features and helps to understand the use cases from the architecture perspective.

Among these tools, BentoML stands out for its robust architecture designed specifically for model packaging and deployment. The logic behind BentoML revolves around creating a "Bento"

service that encapsulates the trained model alongside with its dependencies. This service can be easily deployed as REST API or gRPC endpoint, allowing for direct integration into applications.

The key insights are as following:

1. Deployment Approach: BentoML uses containerization to simplify the deployment, while other platforms like Kubeflow and TFX use Kubernetes [12] for orchestration purposes. SageMaker offers a managed service approach when multiple user-friendly options are available.
2. Supported Libraries: BentoML supports wide range of tools including TensorFlow, PyTorch, and Scikit-learn, making it versatile for different types of ML projects. In contrast, cloud tools like SageMaker support various frameworks, but do not explicitly mention them. TFX platform is designed specifically for TensorFlow [17], and this may limit its applicability in projects that use other ML libraries.
3. Scalability: Both Kubeflow and TFX platforms provide exceptional scalability options due to their Kubernetes-based architecture, making them suitable for large-scale ML operations [15]. It's worth noting that scalability comes with increased complexity in setup and maintenance.
4. Ease of Use: BentoML is noted for its user-friendly API, which is beneficial for developers looking for simplicity. In comparison, Kubeflow platform has a steeper learning curve due to its comprehensive features. This trade-off between ease of use and feature richness is a crucial consideration for teams choosing an MLOps tool.
5. Focus Areas: Each tool has its unique attention points. BentoML is primarily aimed at model serving and deployment, while MLflow emphasizes on tracking and registry capabilities [16]. Kubeflow covers the entire MLOps lifecycle, making it a more holistic solution [15]. The choice of tool often depends on the specific needs of the project and the existing infrastructure.

The core architecture consists of several components: the Model Registry for managing model versions, the API Server for serving predictions, and Docker module for containerization support. This modular design enables seamless scaling and management of machine learning models in production. Additionally, BentoML's architecture includes features for model versioning, allowing for easy rollback and A/B testing of different model versions.

Table 4
Machine learning platforms and model deployment characteristics

Inference Platform	Deployment Approach	Supported Libraries	Scalability	Ease of Use	Focus
BentoML	Containers	TensorFlow, PyTorch, scikit-learn	High	User-friendly API	Model serving and deployment
Mlflow	Serverless, Containers	TensorFlow, PyTorch	High	Moderate complexity	Tracking and Registry
Kubeflow	Kubernetes	TensorFlow, PyTorch, MXNet	Very High	Steep learning curve	Full MLOps Lifecycle
SageMaker	Managed Service	Various	High	User-friendly	Model building and deployment
TFX	Kubernetes Managed	TensorFlow	Very High	Moderate complexity	End-to-end ML pipeline

10. BentoML architecture overview

The logic behind BentoML architecture is centered on simplifying the deployment process while maintaining many flexibility options. By packaging models into a single service unit, BentoML reduces the complexity associated with deployment of machine learning models. The service can

be defined using simple Python decorators, allowing the data scientists to focus on model development rather than deployment intricacies.

When users train a model using popular ML libraries such as TensorFlow or PyTorch, they can create a Bento Service by defining an inference function and specifying input/output types. This service can then be serialized and stored in the Model Registry for future use. The Model Registry not only stores the model but also maintains metadata about the model's performance, training data, and hyperparameters, facilitating reproducibility and traceability.

The BentoML architecture consists of several key components [11]:

1. Bento Service: The fundamental unit in BentoML is Bento Service, which encapsulates a trained ML model along with its inference logic and dependencies. Such service can be easily deployed as a REST API or even gRPC endpoint. The Bento Service also includes pre-processing and post-processing logic, ensuring that data transforms are consistent between training and inference.
2. Model Registry: BentoML includes a Model Registry that manages different versions of models. This feature allows teams to track model lineage and facilitates rollback to previous versions if necessary. The registry also supports tagging and metadata management, thus simplifying the identification of models for specific use cases or experiments.
3. API Server: The server exposes the model's inference capabilities through standardized HTTP endpoints, making it accessible for client applications. It handles request parsing, input validation and error handling, providing a robust interface for model serving.
4. Containerization: BentoML supports containerization through Docker, enabling users to create portable images that encapsulate the entire environment required to run the model. This includes not just the model itself, but also all dependencies, ensuring consistency and reliability across different deployment environments.
5. Deployment Options: The users can deploy their Bento Services on various platforms, including cloud services like AWS, Azure or Google Cloud. In addition, this can be done on Kubernetes clusters using the orchestration tools like Yatai. As an expandable and versatile tool BentoML supports edge deployment for IoT devices and mobile applications.
6. Monitoring and Logging: BentoML integrates with popular monitoring instruments to provide the insights into model performance, HTTP request latency and resource utilization. This component is crucial for maintaining model health and detecting drifts in production environment.
7. Adaptive Batching: To optimize the performance BentoML implements adaptive micro-batching that dynamically adjusts batch sizes depending on incoming request patterns and available computing resources. This feature significantly improves throughput for high-volume services.

The following entity diagram illustrates the architecture described above:

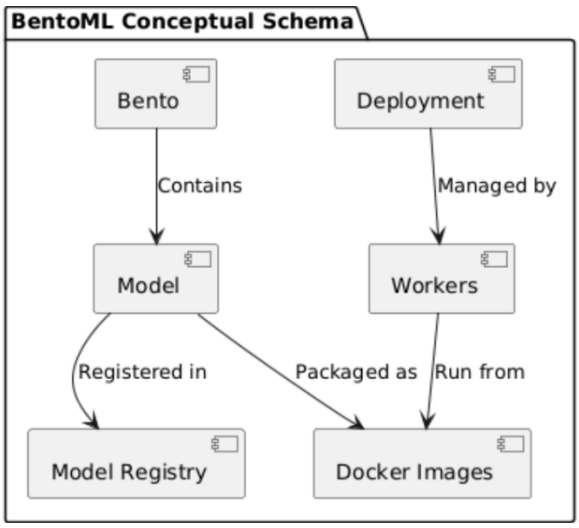

Figure 7: The conceptual scheme of BentoML architecture and principles.

This architecture provides a comprehensive solution for model deployment, addressing the key challenges in MLOps such as versioning, scalability, and integration with existing infrastructure. By abstracting away many of the complexities of deployment, BentoML allows data scientists and engineers to focus on model development and improvement, ultimately accelerating the lifecycle.

11. Deployment on the scale with Yatai

Yatai, an advanced platform developed by BentoML is designed for seamless model deployment on Kubernetes. Its architecture integrates basic principles of scalability, modularity and flexibility, making it highly efficient for managing complex machine learning workloads in production [18]. Yatai leverages Kubernetes autoscaling capabilities to dynamically allocate resources, ensuring that models run efficiently in many diverse computational environments, for example when GPUs are used for inference, and CPUs – just for preprocessing.

One of the fundamental architectural principles of Yatai is its support for microservice-based deployments [13]. Each model component is containerized, thus enabling modular deployment and maintenance. This microservice architecture allows for independent scaling of different parts of the system, optimizing the resource usage depending on workload intensity. For instance, inference workloads may scale on demand with GPU-based microservices, while preprocessing tasks can still rely on CPU-based services.

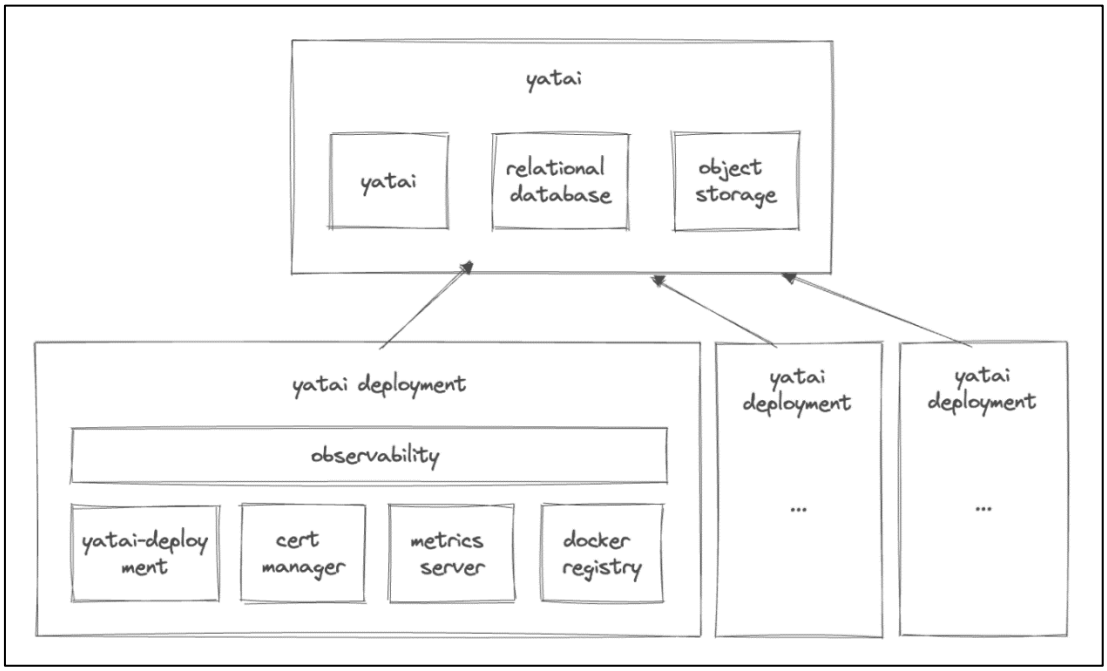

Figure 8: The logic and component relationships of Yatai deployment.

Another significant principle is its support for version control and model lifecycle management. Yatai stores different versions of machine learning models, ensuring seamless rollbacks or updates, making it easier to manage the production environments. These versioning capabilities are tightly integrated with BentoML framework for model hosting. Apparently, this feature streamlines the continuous integration and continuous deployment (CI/CD) workflows. Yatai incorporates many observability features, offering detailed logging, monitoring and tracing capabilities to ensure that models perform as expected in production [18]. This includes:

- Real-time performance metrics
- Resource utilization tracking
- Automated alerting for anomalies
- A/B testing capabilities for model comparison

In addition, the Yatai architecture supports advanced techniques like adaptive micro-batching. This approach improves throughput by batching the requests dynamically depending on system load. This feature is particularly useful for models serving the real-time predictions as it balances response time with computational efficiency. Yatai provides built-in tools for rolling deployments, so that updates can be pushed without the downtime. Also, it provides integration with cloud platforms, offering flexible infrastructure support. Here are some additional features covered:

- Canary deployments for gradual rollouts and zero-downtime updates
- Blue-green deployments for capability to revert the version
- Traffic splitting for controlled experiments

By leveraging Kubernetes orchestration and BentoML model serving capabilities Yatai makes the deployment of machine learning models simple, scalable and more efficient. Its architecture is purpose-built to handle the complexity of machine learning workflows while maintaining high operational efficiency, especially in production environments.

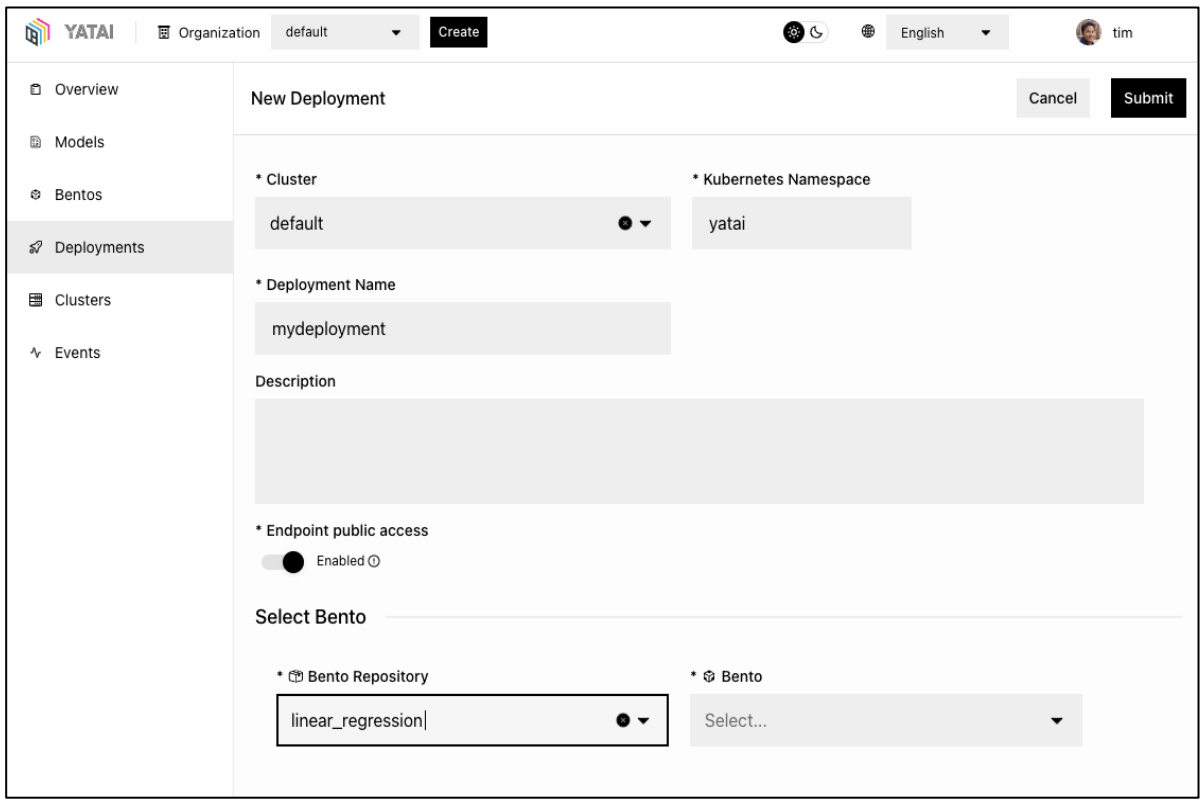

Figure 9: The example of new deployment creation step in Yatai.

12. Prediction model deployment

As a first step, we need to load the trained model serialized in .onnx format into local models registry. In order to simplify the parameter configuration the Config class has been introduced. This class can store various settings such as model path, API endpoints and environment variables.

```
onnx_model = onnx.load("models/split-histo.onnx")
bento_model = bentoml.models.get(CONFIG.MODEL)
bentoml.onnx.save_model(CONFIG.MODEL, onnx_model)
```

It's important to note that the choice of ONNX (Open Neural Network Exchange) format allows the interoperability between different deep learning frameworks, enhancing the model portability.

As a next step in BentoML architecture, we need to define a service class and deploy it to runners. This approach encapsulates the model and its inference logic into a deployable unit.

```
self._runner = bentoml.onnx.get(model).to_runner()
self._service = bentoml.Service(service, runners=[self._runner])
self._service.api(input=NumpyNdarray(), output=JSON())(self.predict)
```

The use of runners allows for efficient resource allocation and parallel processing of requests, which is crucial for handling high-volume prediction tasks. In order to handle the forecasting requests in real-time a RESTful API service can be developed using Python-based technology FastAPI. This framework is chosen for its simplicity and speed, making it ideal for serving machine learning models in production. FastAPI offers remarkable performance due to its asynchronous features and effective request management. The documentation is generated automatically, which reduces the development time and increases API comprehension. The API is expected to receive POST requests with relevant market data from the client application, it interacts with preloaded forecasting model and returns the predictions in real-time.

```
app = FastAPI()
app.include_router(dummy_router)
predictor = Predictor(CONFIG.SERVICE, CONFIG.MODEL)
predictor._service.mount_asgi_app(app)
svc = predictor._service # Entry point for the bentofile
```

The service is configured to handle multiple concurrent requests, providing energy traders and market analysts with near-instantaneous predictions. The asynchronous nature of FastAPI request handling loop ensures that the service can handle high demands of live market environments. Also, the API can be extended to include:

- Input validation to ensure data quality
- Rate limiting to prevent service abuse
- Authentication and authorization for secure access
- Caching mechanisms for frequently requested predictions

The key element within BentoML development lifecycle is bentofile.yaml – a configuration file defining packaging methods and input service. It also allows to reference the aforementioned svc object and requirements.txt file with all libraries required by running process.

```
service: "main:svc"
[...]
```

The bentofile.yaml can be customized further to include:

- Environment variables for different deployment stages
- Resource requirements (CPU, memory, GPU)
- Health check endpoints
- Logging and monitoring configurations

The uniform service deployment procedure for different environments can be easily achieved when Bento service is encapsulated in a Docker container. So, the next stage in the deployment pipeline is service containerization. Docker ensures that dependencies like system libraries and environment variables are bundled together, providing consistency between development, testing, and production environments. The BentoML build command allows to create the docker image and push it to local docker registry. This containerized inference model can be used in a simple docker-compose.yml to define the local deployment environment.

```
version: "3.9"
services:
  energizer:
    image: split-histo:latest
    ports:
      - 3000:3000
    restart: on-failure
    networks:
      - energizer
networks:
  energizer:
    name: energizer
```

This docker-compose setup can be enhanced with:

- Volume mounts for persistent storage
- Environment-specific configurations
- Integration with monitoring services
- Load balancing for high-availability setups

Once the container image is created in registry, it can be deployed into Kubernetes cluster using the orchestration platform Yatai. Kubernetes manages the deployment, autoscaling and maintenance of the containers, it ensures that they remain available and responsive. This setup allows the model to handle real-time requests with low latency, making it suitable for high-frequency market predictions. The Kubernetes deployment can be further optimized by the following steps:

- Implementation of horizontal pod autoscaling that can be based on CPU or memory usage
- Configuration of network policies for enhanced security
- Definition of persistent volumes for model storage and caching
- Integration with cloud-native monitoring and logging solutions

As a summary, the electrical energy market forecasting model can be efficiently deployed, scaled and managed when following the workflow described above. This architecture can provide reliable and timely predictions that can support trade decisions in the volatile energy market landscape.

13.Conclusion

It was demonstrated in this article that development of production quality forecasting solution requires multiple steps: data preprocessing and augmentation, selection of input parameters, selection of machine learning algorithm, hyperparameter optimization, model training and serialization, adding of REST API layer, creation of docker image, networking and autoscaling configuration, deployment of the service into Kubernetes cluster. Apparently, this list is not comprehensive.

While some tools like scikit-learn library and BentoML platform used in this research are Python-based, many other tools are cross-platform, this includes Docker and Kubernetes. It is important to note that major software vendors take the interoperability and reliability quite seriously and invest considerable resources into platform-independent solutions like ONNX standard. For instance the winner algorithm HistogramGradientBoosting implemented in Python has equivalent implementation in .NET called LightGBM. It is advantageous to be able to develop the model in one programming language and deploy to environment that is matching better the skill set of infrastructure team.

The forecasting algorithm could still be placed into simple application that can be launched even from the console. What are the benefits of employing complex technology stack that is proposed in the article? The first aspect is the ability to get the forecast on a remote device, this can be another computer, mobile device or a web page. Also, the prediction can be customized for

specific end user. Another important aspect is scalability. The business requirements for the current task are limited to forecasting the trade volume once an hour, but this is just an example. Once the forecasting algorithm is available it is beneficial to leverage it within the enterprise. So, autoscaling and resiliency become important features affecting the company's financial goals.

Another consideration is design and development of such custom forecasting solution. From the perspective of agile project planning the top-down decomposition of implementation tasks is much more productive than starting a development process with unknown stages and many technical challenges. It is helpful when the artifacts that should be passed in technological chain from one stage to another are known and well specified. This information helps to separate implementation tasks and allows to speed up project development with parallel streams.

Acknowledgements

The authors are grateful to the scientists of G. E. Pukhov Institute of Energy Modeling for providing the hourly data on electricity markets in Ukraine.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] C. Shah, A Hands-On Introduction to Machine Learning, 1st. ed., Cambridge University Press, Cambridge, 2023.
[2] Scikit-learn: Machine Learning in Python. URL: https://scikit-learn.org/stable/.
[3] A new model of the electricity market has been launched in Ukraine. URL: https://expro.com.ua/en/tidings/a-new-model-of-the-electricity-market-has-been-launched-in-ukraine.
[4] M. Osińska, M. Kyzym, V. Khaustova, O. Ilyash, T. Salashenko, Does the Ukrainian electricity market correspond to the European model?, Utilities Policy 79 (2022), 1–14. doi: 10.1016/j.jup.2022.101436.
[5] A. Doroshenko, D. Zhora, O. Savchuk, O. Yatsenko, Application of machine learning techniques for forecasting electricity generation and consumption in Ukraine, in: Proceedings of IT&I 2023, 2023, pp. 136–146. URL: https://ceur-ws.org/Vol-3624/Paper_12.pdf.
[6] A. Doroshenko, D. Zhora, V. Haidukevych, Y. Haidukevych, O. Yatsenko, Forecasting Electrical Energy Consumption for 24 Hours Ahead at Country Scale, in: Proceedings of UkrPROG 2024, 2024.
[7] E. Levinson. Three Approaches to Encoding Time Information as Features for ML Models. Nvidia Developer Technical Blog, 2022. https://developer.nvidia.com/blog/three-approaches-to-encoding-time-information-as-features-for-ml-models/.
[8] S. Haykin, Neural networks: a comprehensive foundation, Prentice Hall, 1998.
[9] V. N. Vapnik, Statistical learning theory, Wiley, 1998.
[10] C. M. Bishop, Pattern recognition and machine learning (Information Science and Statistics), Springer, 2006. https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf
[11] Introduction to BentoML. https://docs.bentoml.com/en/latest/get-started/introduction.html.
[12] Kubernetes Cluster Architecture. https://kubernetes.io/docs/concepts/architecture/.
[13] Yatai tool: https://bentoml.com/blog/yatai-10-model-deployment-on-kubernetes-made-easy/.
[14] N. Klingler. ONNX (Open Neural Network Exchange) Explained: A New Paradigm in AI Interoperability, 2023. https://viso.ai/computer-vision/onnx-explained/.
[15] Kubeflow architecture and principles: https://www.kubeflow.org/docs/started/architecture/.
[16] MLFlow introduction: https://mlflow.org/docs/latest/getting-started/index.html.
[17] TFX Guide: https://www.tensorflow.org/tfx/guide.
[18] Yatai key principles: https://docs.yatai.io/en/latest/concepts/architecture.html.