Purrfect Potty: an IoT solution for cats

Lorenzo Balugani
ID 0001039239
lorenzo.balugani2@studio.unibo.it

Andrea D'Arpa
ID 0000983830
andrea.darpa@studio.unibo.it

***Abstract*— Domestic cats have existed for a very long time, and with them the need of keeping the house clean. The main enabling technology for this, the litterbox, allows kittens to do their deeds in a easy to clean environment - but they can be discouraged to use it if it's dirty. This paper explores a novel application of IoT, that aims at improving living conditions and health of our furry friends.**

***Index terms*—Internet of Things, Web of Things, MQTT, REST API, Cats**

I. Introduction

Pets have been with us for a very large chunk of time. From useful helpers in hunting, to companions in our comfortable homes. With improved lifestyle and a healty economy, a market sector specialized for our quadrupede friends was born, and it's a really healty one. As detailed in this article [1], the market for pet-related items has increased by 5.8% in 2022 if compared to 2021, reaching a tall 77 million euro of market share in Italy alone. In the same article, it is shown that around 10 million cats live in our houses, surpassing dogs by more than a million.

This report contains all the information about a low-cost and easy to build smart device for cats, that allows pet owners to be notified when the litterbox is used, hoping that will allow them to better monitor the cat's usage of the litterbox and to mantain it clean. In the following chapters, details about the hardware and software solutions and structure used will be shown, allowing for everyone to understand the project and make one of their own. All the code used is available at [2].

II. Project Architecture

On the hardware side, this project relies on several key components:

- An Arduino MKR1000, an ARM-based microcontroller produced by Arduino, capable of connecting to any WiFi network thanks to its embedded module;
- A kit of 4 load cells, with included PCB and HX711 chip to allow connection with the Arduino;
- A weighing platform, that will sit on top of the 4 load cells and will be placed under the litterbox - in our case, a piece of plywood;
- A DHT11 temperature sensor;
- A computer to run the data proxy service, along with some other tools. In our case, an OVH Virtual Private Server.

On the software side, the project uses:

- A data proxy application written in Python, using FastAPI as its REST framework;
- A self-hosted Mosquitto server, to allow MQTT communication between the data proxy and the board;
- A self-hosted InfluxDB instance, to allow data storage;
- Grafana cloud, to allow visualization of the data inside InfluxDB;
- A predictor application written in Python, using Prophet, SKLearn and Survival Analysis to make predictions;
- A telegram bot, used to configure the board through the data proxy and deliver notifications;
- A react frontend, used to configure the board through the data proxy;
- Caddy as reverse proxy.

All the Python libraries used can be found inside the "requirements.txt" file located in the root folder of the repository.

From the list of hardware and software, it's possible to locate 6 main actors:

- **The Arduino board**, that communicates with the data proxy using HTTP and can be configured using MQTT. The device senses for WiFi strenght (RSSI), litterbox usage and temperature;
- **The data proxy**, that receives data from the Arduino board, sends notifications, uploads data to InfluxDB and is able to communicate with the board using MQTT. An user can interact with the data proxy by REST, using the Telegram bot or the React frontend;
- **The predictor(s)**, that gets invoked by the data proxy or by some other timer and predicts the number of usages of the litterbox in the next day, temperature, latency and tries to predict when the litterbox will be used next;

- **The Telegram bot**, that allows configuration and notifications;
- **InfluxDB**, that keeps track of data from the WiFi module (RSSI), the litterbox usage and detected temperature;
- **Grafana**, that shows in a dashboard all the data contained inside the time-series database

The system architecture is as its presented in the following figure:

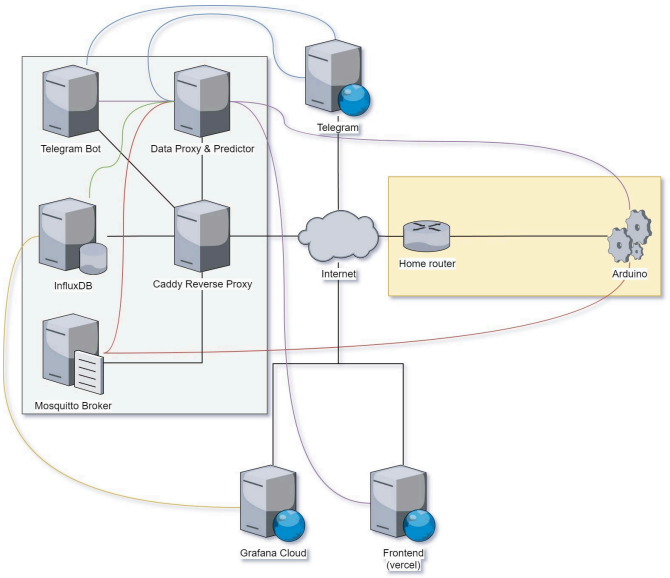

Figure 1: Architecture of the system

The items inside the grey area represent all the services running on the OVH instance, which uses Caddy as its reverse proxy, and the ones in the yellow area represent all the services running inside the residential network. The data proxy can be also placed on the OVH instance, moving everything to the cloud. The black lines represent an hypothetical connection between elements, while the other color coded lines represent:

- Blue: Interaction between items and Telegram server;
- Red: Interaction between items and the MQTT Broker Monquitto on the OVH instance;
- Yellow: Interaction between Grafana Cloud and InfluxDB;
- Purple: Interaction between items and the Data Proxy using HTTP;
- Green: Interaction between the Data Proxy and InfluxDB.

As it can be seen from the diagram, the board communicates with the data proxy using HTTP and MQTT: the first protocol is used to send data from the sensor to the data proxy, while the latter is used to receive configuration changes.

The elements that reside on the OVH VPS are all handled using systemd services, allowing easy monitoring and configuration.

III. Project Implementation

In this chapter, the details about the implementation of the system will be explained in detail. Taking as a reference the IoT pipeline presented in this course, the chapter is organized like so:

- Arduino: data generation, data processing, data communication (from sensor ti gateway);
- Data Proxy: data communication (from gateway to cloud), data analytics;
- InfluxDB: data storage;
- Predictors, Grafana: data analytics;
- Data Proxy Frontend, Telegram Bot: software development;

A. Arduino

The Arduino board serves as this project's data collection system, and in this case is an Arduino MKR1000. The board was chosen because it was already in our possession and we'd probably been better off with an ESP32 with 5V output, since the MKR1000 board has some quirks and issues.

The scketch uses the following libraries:

- WiFi101, allows the usage of the WiFi module;
- PubSubClient, allows the connection with MQTT;
- HX711, allows communication with the HX711 chip, a variable gain amplifier with DAC at 24 bits, that interfaces with the load cells that make up the scale;
- HttpClient, needed to make HTTP requests to the Data Proxy;
- WDTZero, needed to set up a software watchdog;
- FlashStorage, needed to use the flash storage of the board.

As the board gets powered up, the arduino sets up the serial connection and attempts to connect with the WiFi. It then connects to the MQTT broker and subscribes to the topic. Finally, it calibrates the scale and sets up the watchdog. Once the procedure is completed, the built-in led lights up, signaling that the device is ready.

The list of used pins by our sensors, other than the 5V and GND, is as it follows:

- Pin 6, used as the DHT11 data line;
- Pin 5, used as the scale sensor data line;
- Pin 4, used as the scale sensor clock signal;
- Pin 13 (builtin led), used to visually communicate when the board is ready.

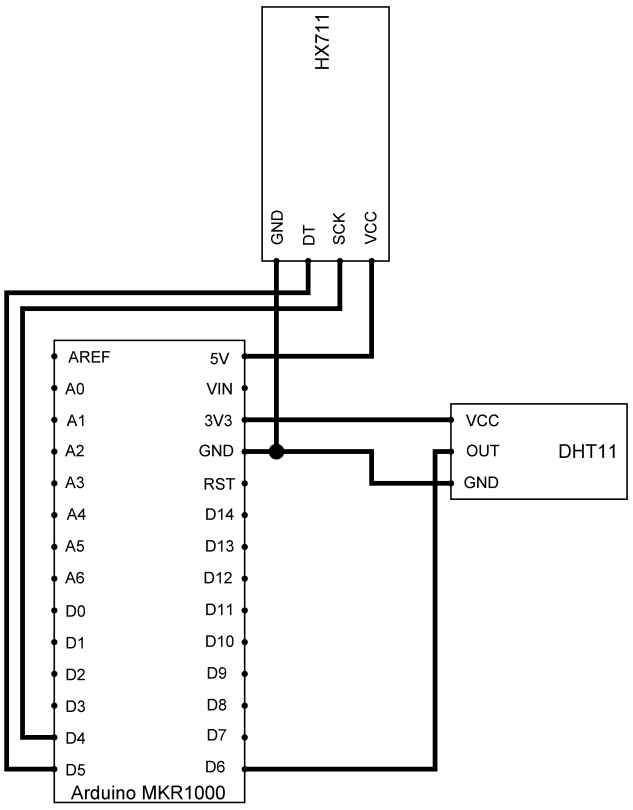

Figure 2: Arduino MKR1000 pinout and circuit diagram

The diagram for the wiring that connects the load cells to the HX711 board can be found at [3].

The Arduino can be configured using the MQTT protocol, and the following variables can be adjusted:

- **sampling_rate**: number of milliseconds to wait till the next cycle;
- **use_offset**: weight (in grams) that once surpassed signals that the litterbox is in use;
- **use_counter**: number of consecutive readings in which the measured weight is greater than use_offset, after which the litterbox is considered in use;
- **tare_timeout**: number of cycles after which the scale self-resets itself while not in use.

These variables get configured by the user through the data proxy, which communicates changes over MQTT.

For each cycle, the Arduino:

1. Checks for new data over MQTT;
2. Waits for [sampling_rate];
3. Resets the watchdog;
4. Reads and sends RSSI data over HTTP;
5. Reads and sends temperature data over HTTP;
6. Gets data from the load cells, and evaluates the status of the scale. If a usage of the litterbox was detected, it sends a HTTP request to the data proxy to alert of its usage;
7. Every 10 HTTP requests, it sends average latency in ms to the data proxy;
8. Loop to 1.

For a more detailed insight of the board's programming, please inspect the provided sketch. At the beginning of this section issues encountered with the board have been mentioned, specifically these two:

1. Random WiFi disconnections, with no real way to reconnect to the network in any way other than resetting the board;
2. Random failure in sending HTTP requests, with no outgoing traffic from the board;

The cause of these problems is not really known. The chance of a memory leak can be crossed out, since available memory in the heap was checked for a long time and lock ups would occur even with lots of free memory available. The most elegant solution we came up with is the beforementioned watchdog, that is able to fully reset the cpu when any of the two situations occur. Looking up online, it seems that this kind of issue is pretty common with this board, which also has some other unfortunate quirks.

In order to handle the plausable chance that a connection lock-up occurs during a litterbox usage, the board saves in flash a flag that will be detected after reboot, forcing the board to notify of the usage even if it didn't happen in the current power cycle. This is not ideal due to wear and tear of memory, but it shouldn't pose as a problem in the short/mid term, since writes occur in an unlikely edge case.

B. Data proxy

The data proxy application is a REST server written in Python, using the FastAPI framework. The data proxy serves as the entity that is reached by the Arduino using the HTTP protocol, sending data in the JSON format. All requests made by the board contain a token field for authentication purposes. Data from the board then gets uploaded to InfluxDB using the official InfluxDB library. Additionally, the data proxy connects to the Telegram's API to push notifications. Upon "litterbox_usage" action, the data proxy starts the prediction module in order to make a prediction of the next time the event will happen again. Alongside litterbox usages, the data proxy receives RSSI, temperature and latency information.

The proxy allows the authentication of an admin user to manage the board settings, detailed in the previous section. Once the changes are made, they are saved locally in JSON format and then transmitted over MQTT to the board, thanks to the Paho library.

In order for the data proxy to run, some environment variables need to be set:

- ADMIN_PASSWORD: the administrator's account password;
- ADMIN_USERNAME: the administrator's account username;
- BOT_TOKEN: the Telegram's bot token;
- CHAT_ID: the Telegram's chat id;
- CORS_ALLOW_ORIGINS: details from which origins CORS requests may come in (if unsure, leave *);
- IFD_BUCKET: InfluxDB bucket name;
- IFD_ORG: InfluxDB organization name;
- IFD_TOKEN: InfluxDB token;
- IFD_URL: InfluxDB instance URL;
- IS_WEB_HOST: address on which the server will be hosted (leave 0.0.0.0 if unsure);
- IS_WEB_PORT: port on which the server will be hosted;
- JWT_KEY: encryption key for JWTs;
- MQTT_BROKER: address of the Mosquitto instance - or any other kind of MQTT broker;
- MQTT_PASSWORD: the password for the MQTT broker;
- MQTT_USERNAME: the username for the MQTT broker;
- THING_TOKEN: token for authentication with the board.

Information about the available routes, schemas, etc. is available at the "/docs" route of the backend, served by the Swagger frontend.

C. Data proxy frontend

The data proxy frontend is a React web application, and serves as a graphical user interface for configuring the Arduino board. The frontend is not tied to any specific backend address, hence it needs to be specified while connecting. Let's have [address] as the base address of the frontend: in order to connect to a backend which address is 127.0.0.1:8000, the address to visit is [address]/127.0.0.1:8000.

D. Predictors

The predictors module is a simple set of functions that uses data gathered from the litterbox's sensor, manipulate them and then apply a machine learning algorithm to do a prediction. The analyzed and then predicted datas are: Usage count in a day-span, temperature and next usage.

Being able to detect how many usages or temperature changes occur during a 24-hour time span provides the possibility to make predictions about the number of interactions with the litterbox that will occur the next day or the registered temperature. There are two separate ways to implement the predictor:

- Linear Regression using the sci-kit-learn Python library;
- Prophet, an additive regression model that works with timeseries with high seasonability.

Daily usages and temperature have both shown to be stationary thanks to the ADF test, so data can be processed without any transfromation.

Linear regression (from the sci-kit-learn library) seems suitable for the task since we are attempting to predict the relationship between two variables: the number of uses (our dependent or target variable) and time (our independent or explanatory variable). Thus, with the dataset containing data gathered over a reasonable number of days for training, the result is a model capable of predicting future events. For the litterbox usage prediction use case, the choice to re-train the model at each prediction cycle stems from the variability in the pet's regular intestinal function. Additionally, the operation is very lightweight, given the small size of our dataset (even when using an entire year of data, we remain in the sub-second range for the operation).

While linear regression is suitable for its simplicity and quick training, Prophet offers advantages like automated handling of seasonality, holiday effects, and irregular data. Prophet adapts to data patterns, simplifies modeling, and provides informations through trend and seasonal changes. Also using prophet, for the litterbox usage prediction, retraining the model seems to be better (also due to changing pet behavior). In the end, temperature is predicted by Prophet, and litterbox usages are predicted by Prophet and by a custom regressor using sci-kit-learn.

In order to attempt to predict when the next usage of the litterbox will occur, we tried survival analysis using the python library "lifelines" that provides a robust framework to experiment in this branch of statistics. A new prediction is provided every time the data proxy receives a request on the litterbox usage endpoint, and it provides a most likely time - along with an upper and lower bound - which are then sent to the user via the Telegram bot. To run the analysis, the problem was modeled by treating each different litterbox usage as a separate instance and by calculating the elapsed time since last litterbox usage.

In order to start the prediction module every 6 hours (or once a day for next day total litterbox usages) systemd timers were used.

E. Telegram Bot

One of the interfaces chosen and implemented to communicate with the user is Telegram. Using a python library called "telebot", it is possible to use the telegram API to develop a chatbot that makes possible to get and send information to our backend service.

The Telegram bot has a custom button interface through which the user can interact. Each button corresponds to a command.There are two types of commands: retrieving information and setting litter parameters.When retrieving information, the bot makes a request to the database and retrieves the desired information (such as the current value of a parameter). When we want to set a parameter, after selecting the parameter itself through the aforementioned button menu, it is sufficient to send a text message containing the integer value to be set for that parameter. Additionally, the Telegram bot sends notifications every time the litter box is used and periodically sends messages to the user with forecasts for both upcoming uses and uses expected for the next day, altough this is done by directly sending API calls to Telegram from the data proxy and predictors.

F. *InfluxDB*

InfluxDB serves as the main storage of all the data produced by the sensor and by the prediction module. Instead of using the cloud version, it was preferred to install an instance of Influx on a OVH VPS, since it was ready at hand. The installation went smoothly, and no issues have been encountered since.

Two different buckets are used: one that contains data about temperature, latency, RSSI and litterbox usage, and another that contains predicted data. Their contents are displayed on the Grafana dashboard.

G. *Grafana*

Grafana serves as the main dashboard of this project, and shows all the relevant data inside the InfluxDB bucket, such as temperature history, litterbox usages, projected litterbox usages and RSSI history. No issues have been encountered since the setup of the dashboard. It displays the temperature (history and latest), the RSSI, latest latency, usage of the litterbox and predicted usages.

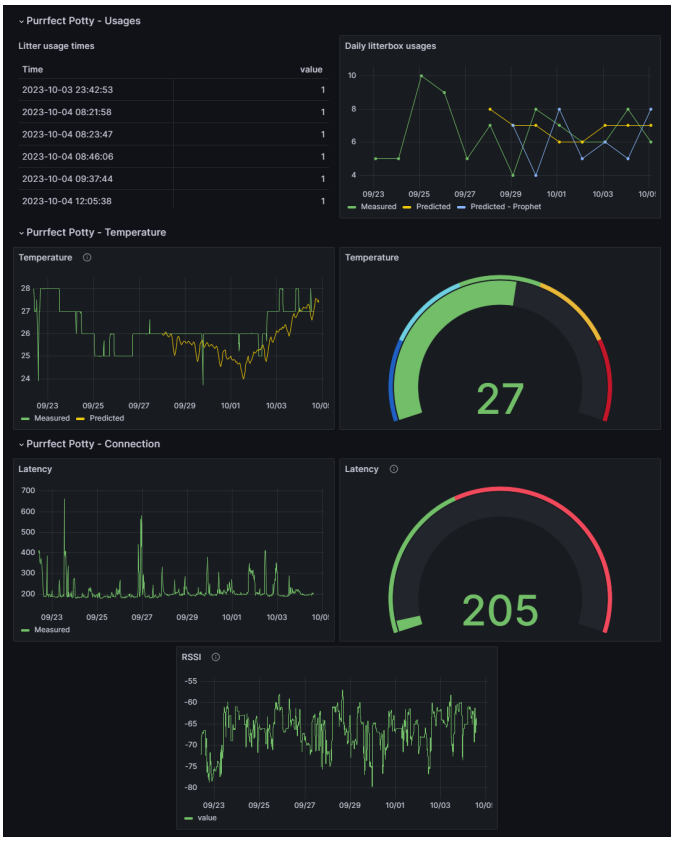

Figure 3: Grafana dashboard

H. *Running the project*

In order to fully run the project, it's mandatory to possess the same hardware that is described in chapter 2. To simply run the software-side of this project, which is available on Github [2], Python 3.9 or greater is required, along with all the libraries contained in the "requirements.txt" file. In order to work, all the environment variables described in chapter 2.B and chapter 2.E need to be set, along with appropriate InfluxDB and Grafana configuration. The data proxy and the telegram bot can be setup as systemd daemons for a long-term setup. The react frontend does not require any setup, as it's already being hosted at address http://purrfectpotty-configurator.iot.fermitech.info/. In order to use it, however, the data proxy must be served over https.

IV. Results

A. *Latency*

The latency of the data acquisition process was measured over a span of 1.000 requests, and it measures the elapsed time from data getting sent from the board, to the data proxy and reaching the influxdb instance. With the described architecture, the results are as it follows:

Stage	Average latency
Board to data proxy	162ms
Data proxy to InfluxDB	43ms
Total latency	205ms

Table 1: Average latency in the data acquisition process

Data was collected by measuring average latency of HTTP requests from the board to the Data Proxy (which comprehends data upload to InfluxDB), and average latency of InfluxDB API calls made by the Data Proxy for the RSSI and temperature requests. The latency is deemed acceptable, since this is not the kind of application that requires lightning-fast communication.

B. Accuracy of the prediction

In order to evaluate accuracy of the system, we compared forecasted data with the sensed one using the Mean Average Deviation metric, which is defined as $\text{MAD} = \frac{\sum(|\text{actual}-\text{forecast}|)}{n}$, and Mean Square Error metric, which is defined as $\text{MSE} = \frac{\sum\left((\text{actual}-\text{forecast})^2\right)}{n}$. The results are as they follow:

Prediction	MAD	MSE
Litter Usage (SKLearn)	1.3846	3.0769
Litter Usage (Prophet)	2.4545	9.1819
Temperature (Prophet)	0.1755	0.3740

Table 2: MAD and MSE values for forecasted values

The results are encouraging, showing how the custom regressor outperforms Prophet in the litterbox usages predictions and how temperature predictions are close enough to the real value.

As for the survuval analysis experiment, which attempts to predict a time range in which the litterbox will get used again, we've counted the number of times the sensor reported usage within the predicted timeframe against total predictions, finding that it was right only 38% of the times. This is result is not great, and the reason for this may come from the impredictability of the phenomena we're trying to predict, which is closely related to the impredictability of the actor's behaviour, in this case a cat.

C. Conclusions

Wrapping all up, this report detailed the realization of an IoT device aimed at cat owners that want to monitor their cat's activity. Results from the predictions is satisfactory, if the survival analisys result is ignored, and the implementation has shown to be reliable during the month of operation. The device has been instrumental, in particular, in monitoring a cat with motory deficiencies that prevent him sometimes to use the litterbox properly. This said, this project can still grow and improve, as its detailed in the next section.

D. Future works and improvements

The project can be improved in two key ways:

- Attempt other methods to predict time of next litterbox usage;
- Swap the Arduino MKR1000 with something that has less hardware quirks and issues and costs less, such as an ESP32 with 5V output.

References

[1] Alessandro Sala, "Pet economy cani & co. Dentro il boom miliardario. Il caso delle lettiere."

[2] Lorenzo Balugani - Andrea Arpa, "Purrfect potty repository." [Online]. Available: https://github.com/Nemesis-FT/PurrfectPotty

[3] Indrek Luuk, "50kg load cells with Hx711 and arduino. 4x, 2x, 1x diagrams. [Online]. Available: https://circuitjournal.com/50kg-load-cells-with-HX711