GENERATIVE LARVAE

Arnaudo Baptiste, Akar Oussama, Berthé Zié, Herbeaux Raphael, Sbai Souleiman

ÉCOLE POLYTECHNIQUE
INSTITUT POLYTECHNIQUE DE PARIS
INSTITUT PASTEUR

CONTENTS

ACKNOWLEDGEMENTS

We wish to express our deep gratitude to our tutor, Mr. François Laurent, researcher at the Institut Pasteur, for his support throughout this Collective Scientific Project. Research and science involve their share of challenges, but his rigor, his commitment and the quality of his advice were essential to the progress and success of our work.

Our thanks also go to Mr. Alexandre Blanc, researcher at the Institut Pasteur, for his great availability. His attentive listening and his willingness to answer our questions greatly facilitated our appropriation of the concepts of *machine learning*.

Finally, we warmly thank the Ecole polytechnique as well as the Institut Pasteur, and in particular all the people involved in the supervision of this project. We address a special thanks to Mr. Yoan Tardy, our coordinator, as well as to Mr. Jean-Baptiste Masson, director of the laboratory *Decision and Bayesian Computation* for having offered us the opportunity to carry out this first long-term scientific project, which, we are convinced, will not be the last.

ABSTRACT

In this project, we develop a complete pipeline of processing, analysis and generation of behavioral trajectories of *Drosophila melanogaster* larvae, in collaboration with the Institut Pasteur. From massive datasets collected under controlled sensory stimulation, we implemented signal processing methods to clean and standardize the trajectories. An in-depth statistical study made it possible to characterize the effect of various genetic modifications on the observed behaviors, notably via the analysis of transition matrices and the use of the Hellinger distance to find the genetic modification inducing the greatest change.

We then designed and trained several generative models, including a sequential variational autoencoder combined with a *Transformer* architecture, capable of producing plausible synthetic trajectories from latent representations. These models made it possible to simulate different types of larval movements (*run*, *cast*, *hunch*...) and credible dynamic sequences.

Beyond the results obtained, this work trained us in the concrete practices of collaborative scientific research: version control with Git, training on HPC cluster thanks to the Maestro Cluster of the Institut Pasteur, modular structuring of code and rigorous empirical evaluation. The perspectives include the improvement of generative models, the analysis of their internal behavior and the extraction of new biological hypotheses from the learned latent spaces.

1 INTRODUCTION

1.1 Interest of the project

The behavior of an organism facing its environment results from interactions between external factors and its genetic heritage. Our study is mainly based on the analysis of the behavior of the larvae in response to *stimuli* such as light or an air current, and on the modeling of these reactions depending on the genotype. These behavioral analyses make it possible to better understand the links between the environment, the genetic determinants and the observed motor responses.

Within the framework of this project, we attempt to simulate the movement trajectories of larvae from sequences of movements of larvae possessing well-defined genotypes. These simulations have the goal of reproducing realistic tests, in conditions close to experimentation, in order to observe and to predict their behavior in response to certain *stimuli*.

A prior study conducted by the Institut Pasteur and developed in the article *Identifying neural substrates of competitive interactions and sequence transitions during mechanosensory responses in Drosophila* [1] written by several researchers of the institute made it possible to establish a correspondence between certain types of movements and the activation of specific neurons. Thus, it becomes possible, from the observation of the movement, to deduce the underlying neuronal activity in the larva, and consequently, to analyze which neuron is activated depending on a *stimulus*.

The choice of the larva as a biological model rests on the relative simplicity of its nervous system, which makes it an excellent subject of study for behavioral modeling. Managing to faithfully simulate its behavior represents a significant advance in the understanding of neuronal mechanisms, and, in the long term, could contribute to better apprehend the bases of human behavior. We have therefore been tasked with completing an already existing code base, developed within the framework of prior works. This base made it possible to simulate certain aspects of larval behavior, but remained incomplete on several essential points for the fine modeling of the trajectories and for the interpretation of the associated neuronal activity.

1.2 Organization of the work

One of the major challenges of this project was to clearly define the different aspects that compose it as well as the associated objectives: from the understanding of the biological and neuronal mechanisms underlying larval behavior, to the computer implementation of simulation models, passing through the analysis of the generated data. Each of these dimensions mobilizes varied and complementary skills, which led us to structure our organization in a collaborative and efficient way.

We chose to divide the work into several poles: Raphaël and Oussama focused on the analysis of the data collected by the institute; Souleiman and Aziz took charge of their simplification and their filtering; finally, Baptiste focused on the training of predictive models. This organization allowed us to progress in a more targeted way while maintaining a coherent overall vision. We nevertheless took care that each person participate, to varying degrees, in each of the sections of the project in order to guarantee a shared understanding and a global coherence.

Weekly meetings were set up from the first weeks in order to make a regular review on the progress of the project. If these meetings were at the beginning dedicated to the handling of our environment, they quickly structured themselves around an agenda announced by a group coordinator. We also benefited from the support of a member of the Institut Pasteur, Mr. François LAURENT with whom we regularly exchanged by videoconference or by email. His feedback helped us to orient ourselves towards feasible paths taking into account the time that was allotted to us.

That said, this freedom of organization also confronted us with some difficulties. The autonomy left to us on the management of the project sometimes destabilized us, notably when it was a matter of defining ourselves our priorities and our deadlines. Nevertheless, our involvement remained constant, and we knew how to maintain an effective group dynamic until the end of the project.

2
PRESENTATION/STATE OF THE ART

The larva of *Drosophila melanogaster* has for several years constituted a model of choice for the study of neuronal circuits: it employs only about 10,000 neurons, while carrying out complex motor sequences, which offers a precious compromise between behavioral richness and accessibility. The Institut Pasteur exploits this paradigm to decipher the neural bases of neurodegenerative diseases and to test therapeutic candidates at large scale.

2.1 Towards a systemic approach of larval behavior

More than ∼300,000 larvae have been filmed under varied stimulation protocols (light, air puff, optogenetic thermogenesis); these massive recordings made it possible to introduce a *probabilistic* dimension into the study of behavior: facing the same *stimulus*, the larva adopts different actions with its own probability distributions. The arrival of the first complete *connectome* of the larva now opens the way to the direct chaining *neuron* → *action* → *sequence*.

2.2 Classification of actions: from the human expert to deep learning

The experimental protocols rely on a restricted dictionary of actions— *forward crawl*, *backward crawl*, *bend*, *hunch*, *roll*, *stop*. Historically, the detection of these patterns was carried out by ideal rules, quickly put in default as soon as the experimental conditions (genotype, physiological state, speed) deviated from the training data. To reduce this bottleneck, `LarvaTagger` proposes a classifier with *transfer learning* capable of being re-trained on a few hundreds of manual annotations and of integrating, as input, the midline as well as the integral contour of the larva. This flexibility is crucial: the literature highlights the absence of consensus on the thresholds separating, for example, a true *stop* from a micro-pause or a *bend* of modest angle.

Despite these advances, two limits persist:

1. **Heterogeneity of performances.** The classifiers show systematic errors when the experimentation over-represents rare patterns (e.g. *Alzheimer* lines where all the actions are slowed down).

2. **Cost of annotation.** Detecting and labeling manually the minority sequences remains time-consuming; a simple re-balancing of the base is not viable in the long term.

2.3 Generative augmentation of the datasets

To remedy these shortcomings, the DBC team proposes a synthetic augmentation of the data. Two complementary axes are explored.

Bayesian synthesis. A probabilistic formalism generates sequences of actions by drawing their parameters (duration, amplitude, speed) from adapted distributions, then by concatenating segments extracted from real data to create a "Frankenstein" trajectory. The coherence of the trajectories is evaluated *a posteriori* by a classifier, which closes a cycle specification → generation → validatio

2.4 Expected contributions

- **For biology.** An enriched base of realistic trajectories, whose distribution can be constrained *in silico*, will facilitate the detection of subtle phenotypes and the functional interpretation of neuronal circuits.

- **For AI.** The project serves as a test bench for generative methods still little explored in the domain of biomechanical time series, in particular the use of elaborated sequential models such as the *transformers*.

- **For software engineering.** The integration of `LarvaTagger`, of the Bayesian pipeline and of the diffusion models on the *Maestro* cluster provides a complete, reproducible and extensible demonstrator.

In sum, the state of the art reveals a clear need: to go beyond simple discriminant classification to move towards a *generative modeling* of the behavior of the larvae, capable of systematically exploring the space of actions and of bringing biology towards a simulation/validation paradigm.

3
COMPUTING ENVIRONMENT

3.1 HPC Cluster

The Institut Pasteur possesses its own high performance cluster, *Maestro*, containing great computing power. *Maestro* is a new cutting-edge computing cluster, deployed and operated by the HPC Core platform. It is composed of servers based on the AMD Zen 2 architecture, each equipped with two processors of 48 cores, that is 96 cores available for multi-threaded tasks. The servers are interconnected via an HDR100 Infiniband network, high performance and low latency, allowing a smooth direct communication between nodes for the works using MPI.

The majority of the servers have 512 GB of RAM, some, called "bigmem", possess 2 TB. Currently, 29 servers are available in the common partition, while 52 servers belong to a dedicated partition (funded directly by research teams but usable temporarily by all). Other servers, including those equipped with graphics cards, are expected in 2021. Maestro is directly connected to the storage servers Zeus and Sonic (for the home and scratch spaces), ensuring a fast access to the data such as the dataset containing the trajectories of larvae.

At the beginning of our project, we followed a certification imposed by the Institut Pasteur before the use of this tool in order to familiarize ourselves with the functioning of the computing cluster, its mode of utilization, as well as the good practices associated with this infrastructure. This step was essential to understand the norms of usage and to optimize our interactions with the system.

Access to this cluster allowed us to carry out computations much more rapidly than what our personal computers would have permitted. This turned out to be particularly valuable during the training of predictive models, a resource-demanding phase, which required both the generation and the processing of large quantities of data.

3.2 Git as collaborative tool

In parallel with our handling of the cluster, we also used a version control system via Git, a fundamental tool for teamwork in any computer project. Git allows to record the history of modifications of the code, to work on different versions simultaneously thanks to branches, and to ensure an effective coordination between the members of the group.

Concretely, we hosted our repository on a collaborative platform (GitHub), which allowed us to centralize not only the code, but also the documentation, the execution scripts and the configuration files. Each member of the group worked on his own branch in order to avoid any direct interference with the work of the others. Once the modifications were finished, a *merge request* was created to propose the integration of the novelties into the main branch of

the project (*main* or *master*). These merge requests were systematically reread by at least one other member of the group, which allowed us to detect possible errors, to discuss certain approaches, and to maintain a global coherence in the development.

Git also allowed us to solve several critical situations, such as the suppression of important files, where it was necessary to return to a previous functional version, then to correct the organization of the project in a clean way. Without such a tool, these errors could have led to a significant loss of time, even to the unintentional deletion of work.

In addition to its technical utility, Git also pushed us to adopt good development practices, such as the clear versioning of files, the writing of explicit commits, and the rigorous documentation of the evolutions of the project. This contributed to structure our work and to guarantee a better readability of the code for all the members of the group, including those joining a task in progress, for we ourselves continued the project that had been begun by other employees of the institute.

4
DATA PROCESSING

4.1 Frequency filtering of trajectories

Due to the method used to record the trajectories in the IP Hecatonchire database, some of the retrieved trajectories are far too noisy or even unusable. This can be explained by problems at the level of the software `Larvatagger`, by the superposition of two larvae during their movement in the test environment when collecting the data, or by the quality of the image.

In order to ignore these undesirable trajectories, we used a filtering system generally employed in signal processing. Concerning the *frames*, the function defining the larvae being bounded, we can consider it as a periodic function f of period equal to the size of the domain of definition T, developed in Fourier series:

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{i\frac{2\pi n}{T}t}$$

Where the coefficients c_n are given by:

$$c_n = \frac{1}{T}\int_0^T f(t)e^{-i\frac{2\pi n}{T}t}\,dt$$

The energy of the signal over one period is defined by:

$$E = \int_0^T |f(t)|^2\,dt$$

The **Parseval theorem** for Fourier series states that:

$$E = T\sum_{n=-\infty}^{\infty} |c_n|^2$$

Each coefficient c_n therefore represents the contribution of the frequency $\frac{n}{T}$ to the total energy of the signal.

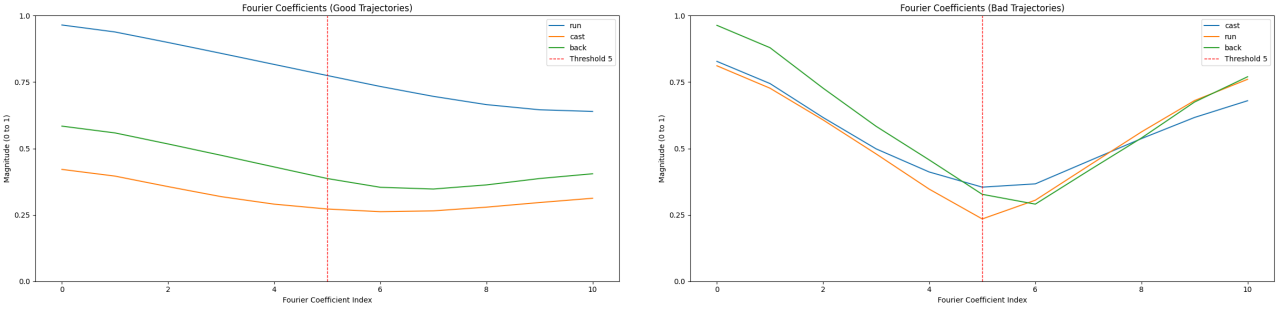
Thus, if the contribution of the high frequencies exceeds a certain threshold, or if their amplitude exceeds a certain threshold, we remove this trajectory from our training data. We chose to remove these trajectories rather than "smooth" them because after a more in-depth study, we discovered that these trajectories represent only about 7% of our data. Therefore, we could do without them.

In the case of sequences, we defined that a sequence is bad if it has a sufficient number of bad *frames*. Thanks to this approach, we were able to extract about 93% of less noisy sequences in order to train our predictive models.

This approach is particularly useful in signal processing to:

- Eliminate noise (parasitic high frequencies)
- Extract slow trends (low-pass filter)
- Isolate fast components (high-pass filter)

The filtering process of the *frames* is illustrated in the figures below.



(a) Good trajectories

(b) Bad trajectories

Figure 1: Average amplitude of the Fourier coefficients for the good and bad trajectories.

Indeed, for the so-called "good" trajectories, the amplitude of the high-frequency harmonics remains weak compared to the first coefficients, reflecting a smoother movement. Conversely, the "bad trajectories" are distinguished by a notable increase in the amplitude of the coefficients associated with high frequencies, which reflects a greater presence of abrupt movements and therefore of noise in the trajectory.

4.2 Regularization of the larva contour

The contours provided by `LarvaTagger` (500 points, many of which are `not a number`) present important fluctuations linked to segmentation errors or to the simultaneous passage of two larvae.
Before learning, it is therefore indispensable to remove this noise in order to prevent the model from making it into a feature.

1. **Discrete Fourier transform.**
   For each coordinate y_n (and, in the same way, for x_n), the FFT is calculated:

$$Y_k = \sum_{n=0}^{N-1} y_n \, e^{-2i\pi kn/N}, \qquad k = 0, \ldots, N-1.$$

2. **Low-pass filtering.**
   Only the first seven harmonics ($k \leq 7$) are kept, the others being set to zero. This empirical threshold eliminates the high frequencies associated with noise while preserving the

general geometry.

3. **Inverse Fourier transform.**
   After filtering, the iFFT restores the smoothed contour $\left(x'_n,\, y'_n\right)$ in real space.

This operation, of complexity $O(N \log N)$, suffices to make the contours regular.

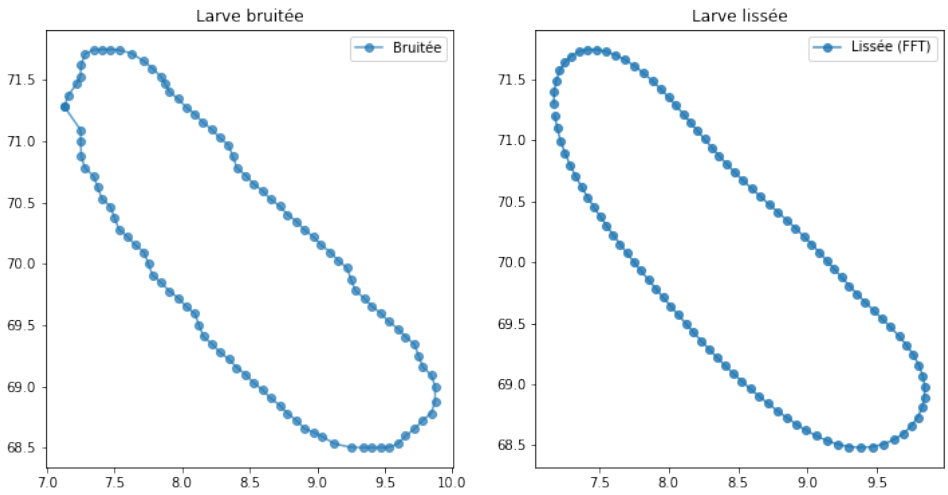


Figure 2: Effect of frequency smoothing: *left*, raw contour; *right*, same contour after keeping the first seven harmonics.

4.3 Simplification of the frames to five vertebra points

To accelerate the processing while keeping the essential information, we tried to reduce each *frame* to five points of the spinal column, then to reconstruct its contour in a smoothed way. The complete procedure is described below.

1. **Selection of five vertebrae:** Five points are taken at regular abscissas along the longitudinal axis of the larva.

2. **Cubic spline interpolation:** A natural spline $(x(t), y(t))$, $t \in [0, 1]$, is fitted to these points, ensuring continuity of the first derivatives.

3. **Computation of normals:** The derivatives $\dot{x}(t)$ and $\dot{y}(t)$ provide the tangent; the unit normal is deduced by rotation of 90°.

4. **Elliptical thickening:** An offset $\Delta(t) = \Delta_{\max}\sqrt{1-4(t-\frac{1}{2})^2}$ enlarges the spline maximally at the center and minimally at the extremities, thus generating the internal and external sides of the "ribbon".

5. **Closure and over-smoothing:** To represent the larva, the obtained polygon is closed, then a periodic spline sampled at 800 points produces the final contour.

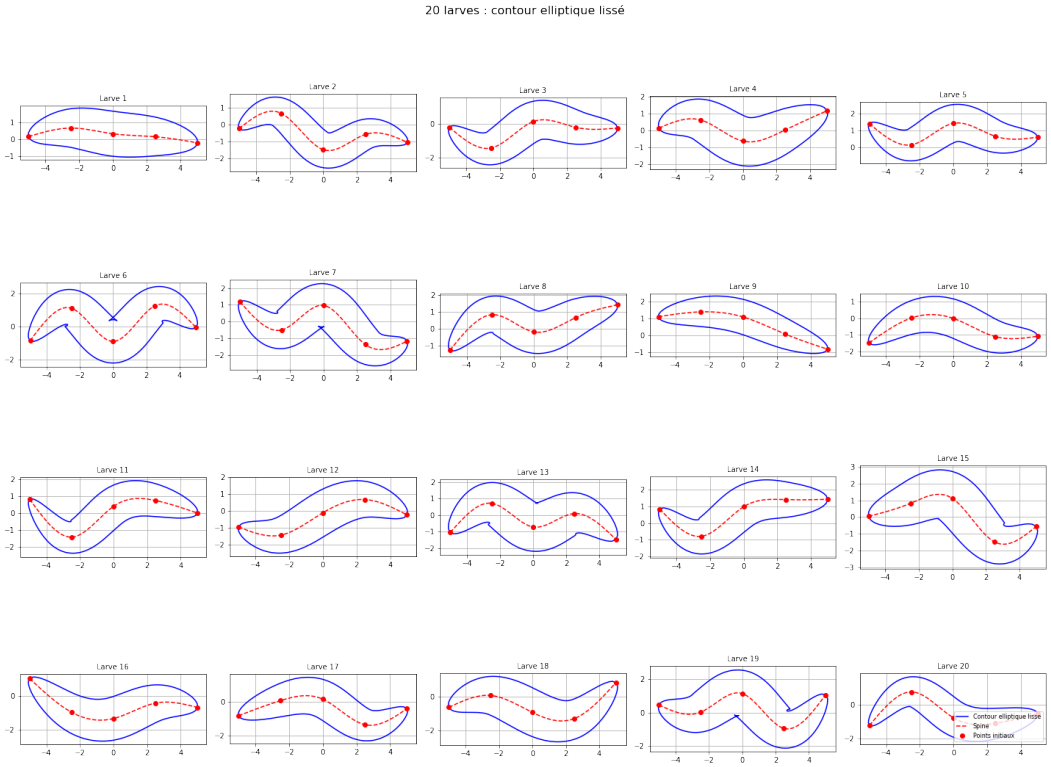


Figure 3: Twenty synthetic contours resulting from five vertebra points each: blue, smoothed elliptical contour; red, vertebral spline; black points, original vertebrae.

Conclusion

The three processing bricks presented — frequency filtering of trajectories, regularization of contours by FFT and compression of the *frames* to five vertebra points — now constitute a coherent chain which transforms a heterogeneous and noisy corpus into a dataset usable by deep learning models.

- The spectral filtering removes **7 %** of trajectories judged irrecoverable and cleans the remaining 93%, guaranteeing the absence of gross discontinuities.
- The FFT smoothing removes the segmentation artifacts while preserving the global geometry, which prevents "learning the noise".
- Finally, the reduction to five vertebrae and the elliptical reconstruction could divide by a factor of ~ 100 the dimension describing each frame; we thus gain in inference speed and numerical stability without sacrificing the relevant morphological measures.

These treatments therefore standardize the representation of the larvae, considerably reduce the data volume and set favorable initial conditions for statistical analysis (section 5) as well as for generative approaches (section 6). The known limitations — empirical choice of spectral thresholds, possible biases introduced by the random selection of vertebrae — will be reevaluated during the training phases; they could be removed by a dynamic adaptation of the parameters or by a supervised learning of the contour profile if necessary. Nevertheless, the current pipeline already provides a robust compromise between biological fidelity, algorithmic simplicity and computational cost.

The avenue of reduction of points having been explored only at the end of the project, we could not use it to reduce the training of the models to the study of only 5 points.

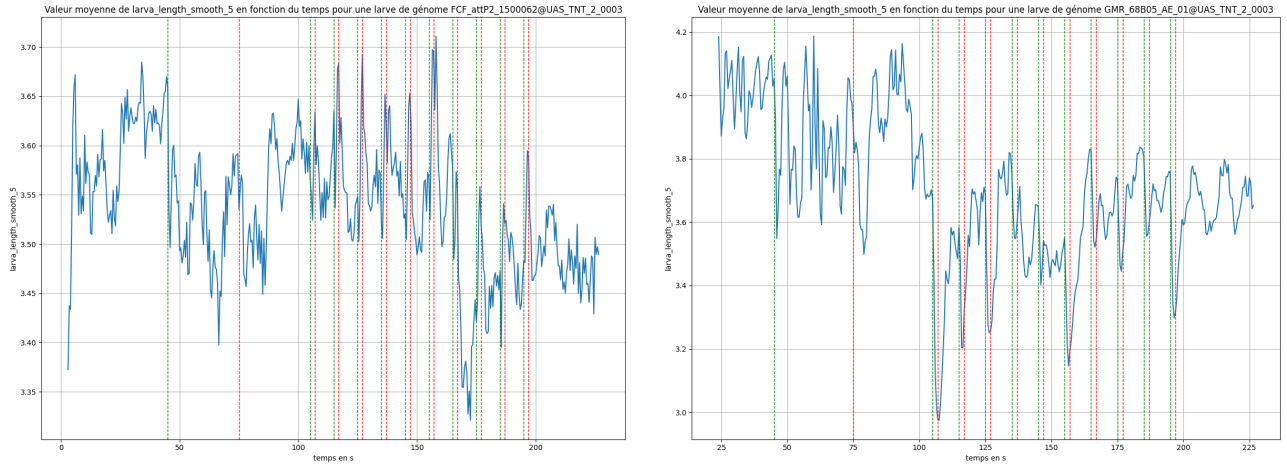
5
STATISTICAL STUDY

One of the important objectives of the Institut Pasteur is to identify the genetic lines that influence the behavior of the larvae according to the *stimuli* they receive. Before generating artificial trajectories, we decided to carry out a statistical study of the data at our disposal.

Thus, we first decided to observe the evolution over time of certain characteristics of the larva at fixed experiment. In this way, we can observe differences in the reactions of the larvae depending on whether their genome has been modified or not.

To do so, we created exploratory graphs to have a first intuition about the genetic lines that could induce very different behaviors compared to the control genome. These graphs represent the average evolution of certain characteristics over time for various experiments. Here is an example below:

5.1 The length of the larvae



(a) Control genome (FCF)

(b) Modified genome (GMR)

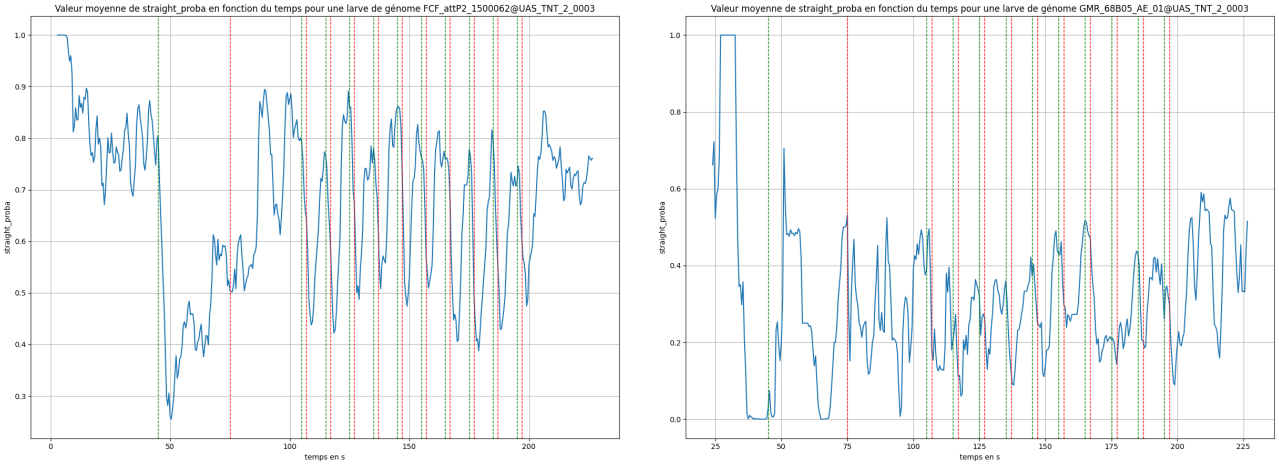
Figure 4: Time evolution of the length of a population of larvae.

Here, the characteristic studied is the length of the larva, the *stimuli* are air puffs (their start is signaled by a vertical green bar and their end by a vertical red bar), the graph on the left corresponds to the control genome while the one on the right corresponds to a modified genome. There is a first long *stimulus* of 30 seconds after 45 seconds of experiment, then 10 short *stimuli* of 2 seconds from 105 seconds, whose beginnings are spaced by 10 seconds.

Already we notice that the larvae with modified genome seem to react more clearly and precisely to the short *stimuli*. Moreover, we observe that the larvae with modified genome

generally have a shorter length than those with the control genome when they are exposed to short-duration *stimuli*.

5.2 Probability of moving forward in a straight line



(a) Control genome (FCF)

(b) Modified genome (GMR)

Figure 5: Time evolution of the probability of moving forward in a straight line.

In this analysis, the characteristic studied is the probability for the larvae to move forward in a straight line, keeping the same protocol as the previous paragraph.

We first observe that, whatever the type of *stimulus*, this probability is on average lower in the larvae with modified genome (GMR) than in those with the control genome. Then, in the control larvae, a systematic decrease of this probability is observed in response to the *stimuli*, whether they are short or long.

However, faced with a prolonged *stimulus*, we notice a rebound of this probability, which suggests the existence of a characteristic time beyond which the larva regains a stationary behavior, even in the continuous presence of the *stimulus*. On the other hand, the larvae with modified genome do not present a systematic response to the *stimuli*, which could indicate an alteration of their adaptation mechanism.

- Conclusion

After comparison on several characteristics and several experiments, it seemed to us that there was a genetic line (GMR) that had a more important influence on the induced behaviors than the others, which was confirmed to us by Mr. LAURENT. However, at this stage, we did not have quantitative results to affirm it with certainty.

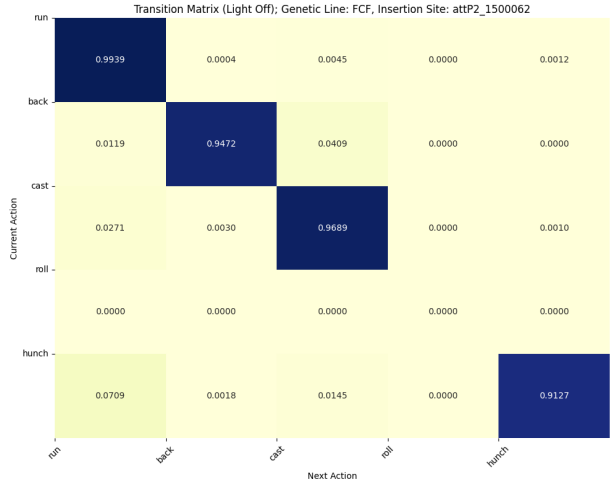
On the advice of Mr. LAURENT, we focused our attention on the response of the larvae to the long *stimuli*, since it would seem that they reach a stationary state from a certain moment.

An idea was therefore to exhibit a characteristic time of evolution between the beginning of the stimulus and the return to the initial behavior.

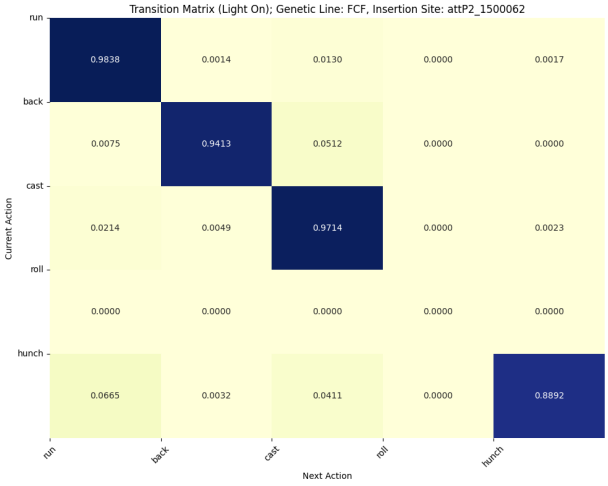
5.3 Change induced by the genetic line

At first, we sought to characterize the motor behaviors of the larvae in response to environmental *stimuli*, in particular variations of light. To do this, we adopted a probabilistic modeling approach using a Markov model in order to represent the different phases of behavior of the larvae as well as the transitions between these states.

Each behavior of the larva was assimilated to a state in the model. By observing the transitions between these states over a given period of time, we were able to construct a transition matrix describing the probabilities of passage from one behavior to another. This matrix was then compared between different experimental conditions (light on versus light off).

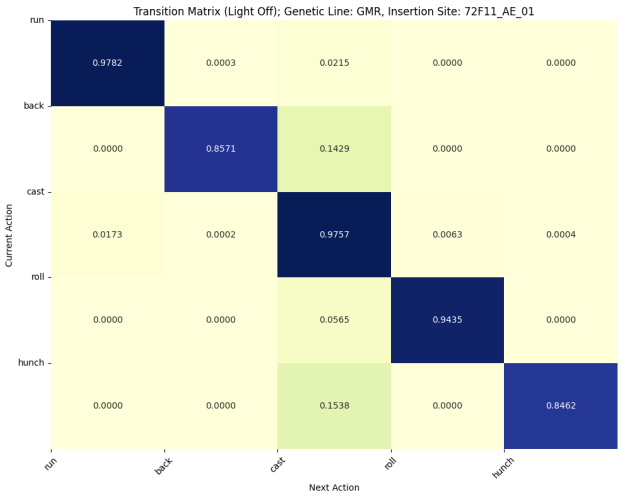


(a) Transition matrix FCF, without light *stimulus*

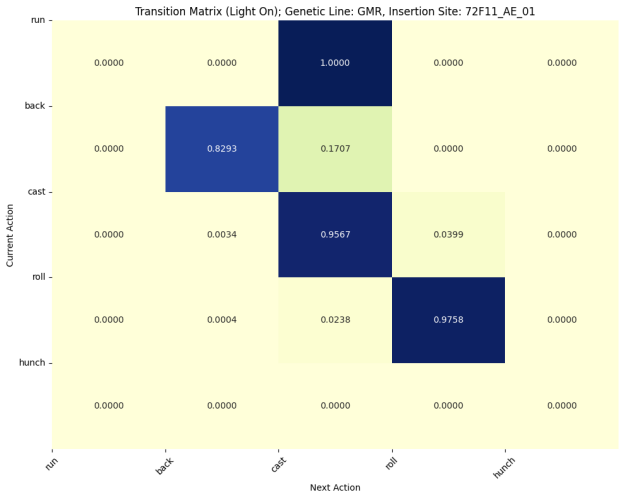


(b) Transition matrix FCF, with light *stimulus*

As the figure above shows, in the **control larvae** (FCF), the sudden activation of the light leads to a **notable increase of the probability of transition to the state of *cast***, whatever the type of previous action. This behavior can be interpreted as a phase of *exploration*, possibly triggered by the appearance of an environment perceived as new or uncertain. This type of response is coherent with what is observed in many animal models, where a sudden sensory change leads to a behavior of analysis of the environment before resumption of normal activity. The increase of the *cast* can therefore be seen as an adaptation strategy, allowing the organism to re-evaluate its environment before continuing a trajectory of movement.



(a) Transition matrix GMR, without light stimulus



(b) Transition matrix GMR, with light stimulus

Conversely, in the **genetically modified larvae** via the expression of the **GMR gene**, we observe a marked and significant effect: the **almost complete disappearance of the phases of *run***, replaced by prolonged sequences of *cast* or of stop. This behavior could reflect a *hypersensitivity to light stimuli* or a *disorder of sensorimotor integration*, preventing the larva from engaging in a coherent locomotion. The absence of *run* could thus reflect a state of vigilance or high caution, the larva preferring to remain in exploration or frozen rather than moving in an environment perceived as potentially threatening or disturbing.

However, it is important to underline certain methodological limits of our approach. Indeed, the Markov model supposes stationary transitions of behaviors in time, which may not be totally representative of the biological reality. For example, it has been observed that the larvae, after a certain time of exposure to a given environment, can become accustomed to it. This habituation leads to a progressive normalization of their behavior, making the differences initially observed less marked. This phenomenon of adaptation must be taken into account in the interpretation of the results, because it could mask or attenuate the effects of the *stimuli* in prolonged temporal analyses.

In conclusion, this behavioral modeling by Markov chains offers a relevant tool to quantify and compare the larval motor strategies according to environmental or genetic factors. It could be enriched by dynamic models integrating the time factor or learning, to better capture the evolutions of the behavior throughout the experiment.

In order to obtain more quantitative results on the influence of the different genetic lines on the attitude of the larvae, an idea was to reuse the transition matrices already mentioned earlier in order to see how "far apart" they were depending on the genome of the larvae.

To be able to compare two genetic lines according to the differences with respect to the control genome that they induce on the transition matrices, it was first necessary to define an appropriate distance. We therefore defined distances on transition matrices which are derived

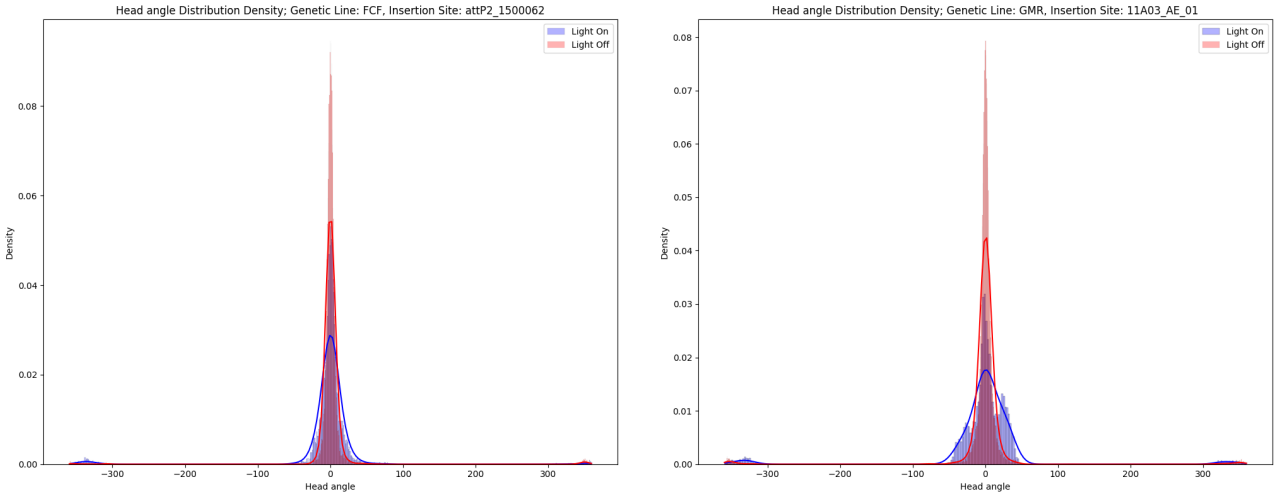
from distances measuring the dissimilarity between probability distributions. Initially, our attention was focused on the three following distances: *Kullback-Leibler* (which is not really a distance because it is not symmetric), *Jensen-Shannon* and *Hellinger*. Finally, for reasons of simplicity, we only kept the *Hellinger* distance, whose formula, for probabilities on finite or countable sets, is:

$$H(P,Q) = \frac{1}{\sqrt{2}}\sqrt{\sum_x \left(\sqrt{P(x)} - \sqrt{Q(x)}\right)^2}$$

To adapt the formula to transition matrices, it is enough to see the matrix as as many probability distributions as there are rows, and one thus carries out a calculation of distance row by row, then one takes the average of the distances obtained.

Equipped with this metric, we were able to explore the database to determine which genetic line induced the greatest change of transition matrix (at fixed experimental protocol). We found that **for the experimental protocol consisting in lighting the larvae twice for 15 seconds with 15 seconds interval between the two lightings, modifying the GMR line at site 72F11 induced a Hellinger distance of** 0.54 **with respect to the transition matrix of the control larvae**, reflecting **substantial differences**.

We therefore decided to plot histograms representing the probability distributions of certain characteristics of the larva (speed, head angle...), depending on the genetic line and the state of the environment in order to observe these differences.



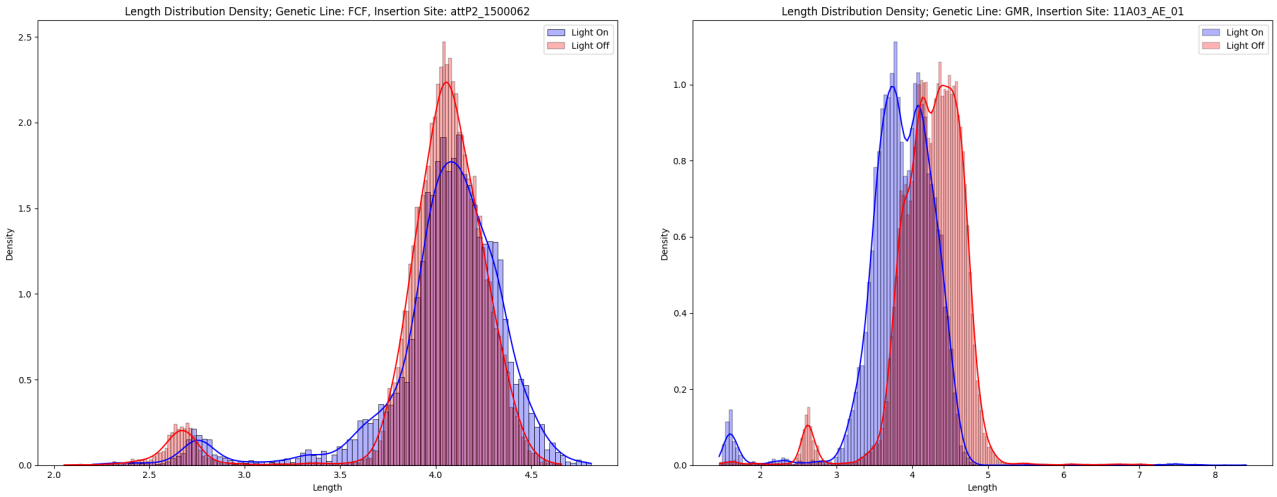
(a) Control genome (FCF)

(b) Modified genome (GMR)

Figure 8: Probability distributions of the head angle of the larva with and without LED.

We see here that the modified genome does not modify the mean value of the head angle but that it seems to increase its variance when the light is on, which seems to confirm that this genome creates larvae more sensitive to *stimuli*.

Here, we even observe that the mean length of the larva is lower when the light is on in the



(a) Control genome (FCF)

(b) Modified genome (GMR)

Figure 9: Probability distributions of the length of the larva with and without LED.

case of the modified genome, which is not the case for the control genome, which reinforces the previous hypothesis.

5.4 Conclusion

This method constitutes an interesting approach to detect and to note the substantial changes induced by the modification of certain genetic lines on the behavior of the larva. However, this approach fails to identify the subtle changes of behavior, like those identified by Mr. Alexandre Blanc in his article *Statistical signature of subtle behavioural changes in large-scale behavioural assays* [2] using refined techniques of statistical learning.

6
TRAJECTORY GENERATION

6.1 Towards a refinement of the transitions between actions

As stated in our intermediate report, a first approach to generating synthetic trajectories was to concatenate pieces of already existing trajectories and to perform the transitions by linearly interpolating the *frames*, while adding Gaussian noise. This method proved unsatisfactory, which led us to adopt a second approach using an **auto-encoder**.

6.1.1 • A new tool: auto-encoders

Auto-encoders are unsupervised neural networks designed to learn a compact and efficient representation of the data. Their main objective is to compress and to reconstruct an input datum, the compressed version of the datum is called latent representation.

An auto-encoder is composed of two main modules:

- **The encoder** f_θ, which projects the input $\mathbf{x} \in \mathbb{R}^N$ into a latent space of reduced dimension $\mathbb{R}^k$ with $k < N$, producing a latent vector $\mathbf{z}$:

$$\mathbf{z} = f_\theta(\mathbf{x})$$

- **The decoder** g_ϕ, which attempts to reconstruct the original input from the latent representation :

$$\hat{\mathbf{x}} = g_\phi(\mathbf{z}) = g_\phi(f_\theta(\mathbf{x}))$$

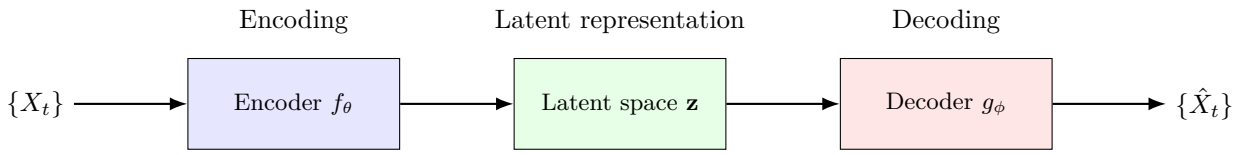


Figure 10: Architecture of a classical auto-encoder

The training of the auto-encoder relies on the minimization of a loss function which measures the gap between the input $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$. A function commonly used is the mean squared error:

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{N}\|\mathbf{x} - \hat{\mathbf{x}}\|^2$$

This formulation allows the network to learn to encode the essential information of the input into a lower-dimensional latent space, while keeping the ability to reconstitute the original

information with a minimum of loss.

Auto-encoders are notably used for dimensionality reduction, anomaly detection, data compression or feature extraction.

6.1.2 • Generation of transitions using an auto-encoder

Once the auto-encoder is trained, the idea consists in exploiting its ability to project the data into a reduced-dimensional latent space, in which the trajectories are represented in a more compact and structured way.

To interpolate between two *frames* $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{end}}$, our first idea was to begin by computing their respective latent representations $\mathbf{z}_{\text{start}} = f_\theta(\mathbf{x}_{\text{start}})$ and $\mathbf{z}_{\text{end}} = f_\theta(\mathbf{x}_{\text{end}})$, then to generate intermediate latent vectors by performing a linear interpolation in the latent space:

$$\mathbf{z}_\alpha = (1-\alpha)\mathbf{z}_{\text{start}} + \alpha\mathbf{z}_{\text{end}}$$

With $\alpha \in \{\frac{k}{m}, k \in [\![1, m-1]\!]\}$ where m is the chosen number of interpolation points.

Each intermediate latent vector $\mathbf{z}_\alpha$ is then passed into the decoder of the auto-encoder to obtain a generated frame $\hat{\mathbf{x}}_\alpha = g_\phi(\mathbf{z}_\alpha)$. We thought that this process would allow to reconstitute progressive and realistic transitions between two observations, which is particularly useful to simulate continuous movements or to complete incomplete sequences. However, we had not realized nor measured the complex geometry that the latent space can have: as the left-hand side of figure 12 shows, it may very well happen that neighboring points give radically different *frames*, or even that one gives an aberrant *frame*!

In order to generate plausible transitions, we therefore had to be clever and train an auto-encoder to interpolate by itself!

Concretely, we trained an auto-encoder to reconstruct a sequence of 10 frames, being given as input only the first and last *frames*. The model thus learned to deduce the intermediate *frames*, which allowed us to generate the transitions presented on this Google Drive.

6.2 Use of generative AI methods

6.2.1 • An elementary brick: variational auto-encoders

As we have seen, the use of auto-encoders proved appropriate to predict the positions of a larva between several *frames*, thus allowing somewhat more realistic interpolations. Nevertheless, this type of model does not truly allow to generate trajectories. During our research, our interest was attracted to a new class of model: the **variational auto-encoders** (cf. figure 11).

Introduced in 2013 in the foundational article *Auto-Encoding Variational Bayes* [3] by Diederik P Kingma and Max Welling, the variational auto-encoders (abbreviated as VAE hereafter) work on the same principle as classical auto-encoders, with the difference that these

encode the data in a probabilistic latent space, associating to each datum a probability distribution. Naturally, the training process of the VAE is guided by the reconstruction of the datum that was given as input. But another aspect of the training of a VAE consists in penalizing the fact that the probabilistic distribution of the latent space strays too far from a distribution given and chosen in advance, called *prior*. The choice of the *prior* is determinant for the geometry of the latent space, and defines the form of the distributions used by the encoder. For example, if the *prior* is a Gaussian $p(z) = \mathcal{N}(0, I; z)$, the probability distributions obtained by *encoding* the data x will be of the form $q_\theta(z|x) = \mathcal{N}(\mu, \text{diag}(\sigma_1^2, \ldots, \sigma_n^2); z)$. To ensure that the q_θ "remain close" to p, the VAE will seek to minimize the **Kullback-Leibler divergence** of q_θ with respect to p, defined by:

$$D_{KL}(q_\theta(z|x)||p(z)) \triangleq \int_z q(z|x) \ln\left(\frac{q(z|x)}{p(z)}\right) dz \in [0, \infty]$$

The interest of using a Gaussian *prior* is that then $D_{KL}(q_\theta||p)$ takes the following form:

$$D_{KL}\left(\mathcal{N}(\mu, \text{diag}(\sigma^2)) \parallel \mathcal{N}(0, I)\right) = \frac{1}{2}\sum_{i=1}^{d}\left(\sigma_i^2 + \mu_i^2 - 1 - \log\sigma_i^2\right)$$

which is easily computable algorithmically.

More precisely, the VAEs with Gaussian *prior* encode the data into a mean μ and a log-variance $\log\sigma^2$ (where the σ_i are the coefficients of the diagonal covariance matrix of the associated Gaussian distribution). Then, the model samples a point of the latent space according to the law $\mathcal{N}(\mu, \sigma^2)$. Only, if this sampling were carried out as is, the propagation of the gradient could not be performed, the operation carried out not being differentiable. That is why the article *Auto-Encoding Variational Bayes* [3] introduces the reparametrization trick which consists in sampling $\varepsilon \sim \mathcal{N}(0, I)$ then computing $z = \mu + \varepsilon \odot \sigma$ (where $\odot$ is the coordinate-wise product) so that $\theta \mapsto z$ is indeed a differentiable function.

Our choice of VAEs was guided by their capacity of generation of new data, itself due to the great regularity of their latent space. Indeed, in the latent space of a classical auto-encoder, the regularity is lower, and the decoding of a point of the space however close to a point corresponding to an encoded datum can give aberrant results as can be seen in the left-hand side of figure 12, where the decoding of a point in the neighborhood of an isolated purple point will give aberrant results). In the case of a VAE, and if the training has gone well, the generation of new data is made possible by decoding an element of the latent space sampled according to the chosen *prior*. We therefore tried to reproduce this technique in our generative models.

6.2.2 • Use of VAEs in our models

Our goal being to generate **trajectories**, that is to say formally sequences of vectors (representing *frames*), we had to find a way to encode and to decode these sequences. Since these could be of variable lengths (and being relatively complex), we could not simply use a multilayer perceptron which would take as argument the flattened sequence of vectors, encode it and try

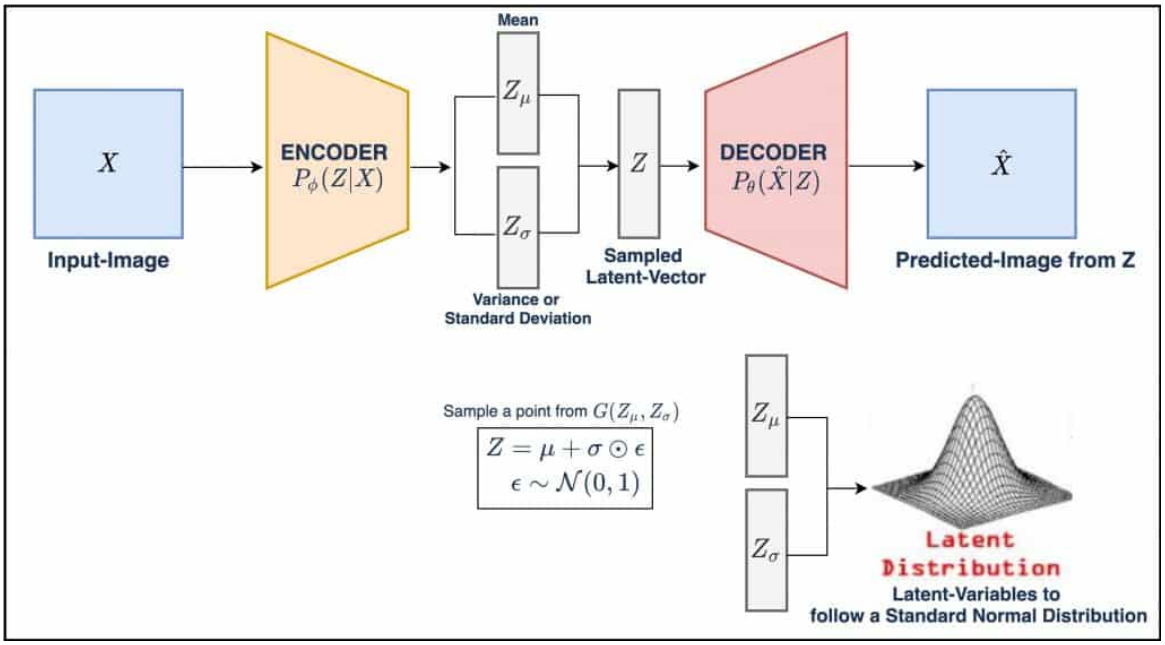
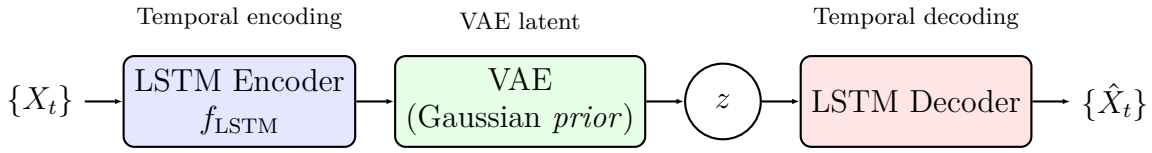


Figure 11: Architecture of a VAE showing the reparametrization trick (source: Kaggle)

to reconstruct it.

We used sequential models in order to capture the spatio-temporal dependencies between the *frames* of our sequences, whose number could be variable. The idea is first to encode the sequence of *frames* into a fixed-size vector using a sequential model, then to encode and decode this fixed-size vector using a VAE, then to reconstruct the initial sequence using a sequential model.

As stated in our intermediate report, our first model was constituted of an LSTM, whose encoding function will be denoted f_{LSTM} and whose hidden state corresponds to the encoding of the sequence and captures the dependencies, notably temporal, between the *frames*. With the aim of generating trajectories, we placed in series with this LSTM a VAE with Gaussian *prior*, as well as an LSTM whose goal was to decode the output of the VAE and reconstruct the sequence passed as input.



Since we replaced our classical auto-encoder with a variational auto-encoder, the loss function used to train our model becomes:

$$\mathcal{L} : (X, \hat{X}) \mapsto \frac{1}{Nl} \sum_{i=1}^{l} ||X_i - \hat{X}_i||_2^2 + D_{KL}(q_\theta(z|f_{\text{enc}}(X))||p(z))$$

where $X \in (\mathbb{R}^N)^l$ is the input sequence (formally: a sequence of l vectors of size N the num-

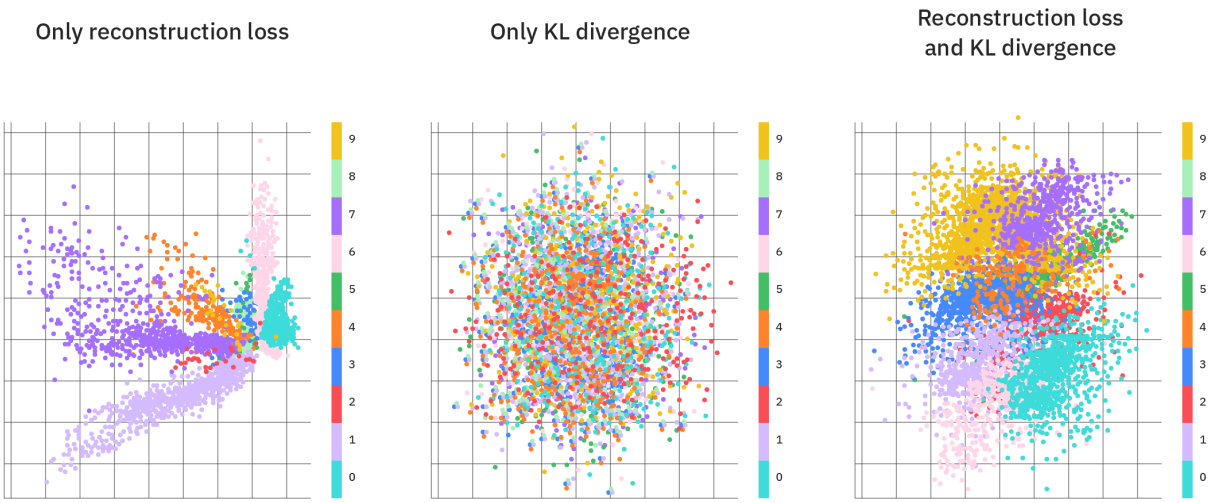


Figure 12: Geometries of latent spaces according to the chosen loss function (source: IBM)

ber of *features*, the length of such sequences being able to vary from one sequence to another), where $\hat{X}$ is the reconstructed sequence and where f_{enc} is the encoding function.

We will not develop this model further because we did not devote much time to it, preferring to replace the LSTM with a **transformer** which truly constitutes the state of the art in terms of sequential model.

Introduced in the article *Attention is all you need* [4] by the Google Brain team, the *transformers* constitute a true revolution in the processing of sequential data (in particular language, but not only) and this for several reasons. Indeed, their mechanism of *attention* allows a larger memory and a greater understanding of context compared to LSTMs, which we could observe ourselves: when we tried to generate relatively long trajectories (100 *frames* and more), we noticed that our LSTM "derailed" as the generation progressed, finally generating aberrant *frames*. Moreover, the parallel architecture of the *transformers* allows a parallelization of the computations on graphics cards, and therefore a faster training.

Let us explain quickly how the mechanism of *self-attention* works in the framework of our model dealing with trajectories of larvae. Given a trajectory $X = (X_1, \dots, X_l) \in (\mathbb{R}^N)^l$, the objective of the encoder of the *transformer* is to provide a sequence $Z = (z_1, \dots, z_l) \in (\mathbb{R}^m)^l$ (where m is a hyper-parameter, that is to say a parameter fixed before training), where each z_i is a **contextualized summary** of the position i knowing the whole sequence. To do this, for each *frame* X_i, the model computes:

- The *query* $q_i = W_Q X_i$ where $W_Q \in \mathcal{M}_{d_k,N}$
- The *key* $k_i = W_K X_i$ where $W_K \in \mathcal{M}_{d_k,N}$
- The *value* $v_i = W_V X_i$ where $W_V \in \mathcal{M}_{d_k,N}$

Once this is done, one computes the matrix of normalized scores $\left(\frac{q_i k_j}{\sqrt{d_k}}\right)_{1\leq i,j\leq l} \triangleq (s_{i,j})_{1\leq i,j\leq l}$ to which one applies a *softmax* on each row to obtain the **attention weights**:

$$\forall i,j \in \{1,\dots,l\}, \quad \alpha_{i,j} = \frac{e^{s_{i,j}}}{\sum_{j'=1}^{l} e^{s_{i,j'}}}$$

Then, z_i is given by:

$$z_i = \sum_{j=1}^{l} \alpha_{i,j} v_j$$

We see that each $\alpha_{i,j}$ expresses the importance of the value v_j in the representation of the *frame* X_i through the vector z_i. Thus understanding how the model computes the attention can make it possible to identify the most important and explanatory components of a trajectory.

In reality, and as we did in our model, each *transformer* has several "attention heads", proceeding in the same way as described above and in parallel, the intuition behind this being that each head specializes in one aspect of the information conveyed by the sequence. The outputs of each head are then concatenated. Moreover, the total device is generally repeated several times, and contains normalization layers allowing to stabilize the training (see figure 13).

We can now explain in more detail the model used whose architecture is represented in figure 14.

- At the beginning of the model, a linear layer projects each *frame* (vector of size $N = 24$) into a space of dimension $d_{\text{model}} = 256$ in order to increase the number of possible attention axes. We also add to each *frame* a classical sinusoidal positional encoding as in the foundational article of the *transformers*, allowing the model to situate each *frame* in the sequence.
- Then, the sequence thus obtained passes through $n_{\text{enc}} = 5$ *transformers* (class `TransformerEncoderLa` of `PyTorch`) with $n_{\text{head}} = 8$ attention heads, providing a sequence $(Z_1,\dots,Z_l)$, l being the length of the sequence passed as input.
- One then performs a *temporal mean-pool* by computing the vector $Z = \frac{1}{l}\sum_{i=1}^{l} Z_i$. With the help of a VAE with Gaussian *prior*, one then encodes this vector into a mean μ and a log-variance $\log \sigma^2$ of size $d_{\text{hidden}} = 64$.

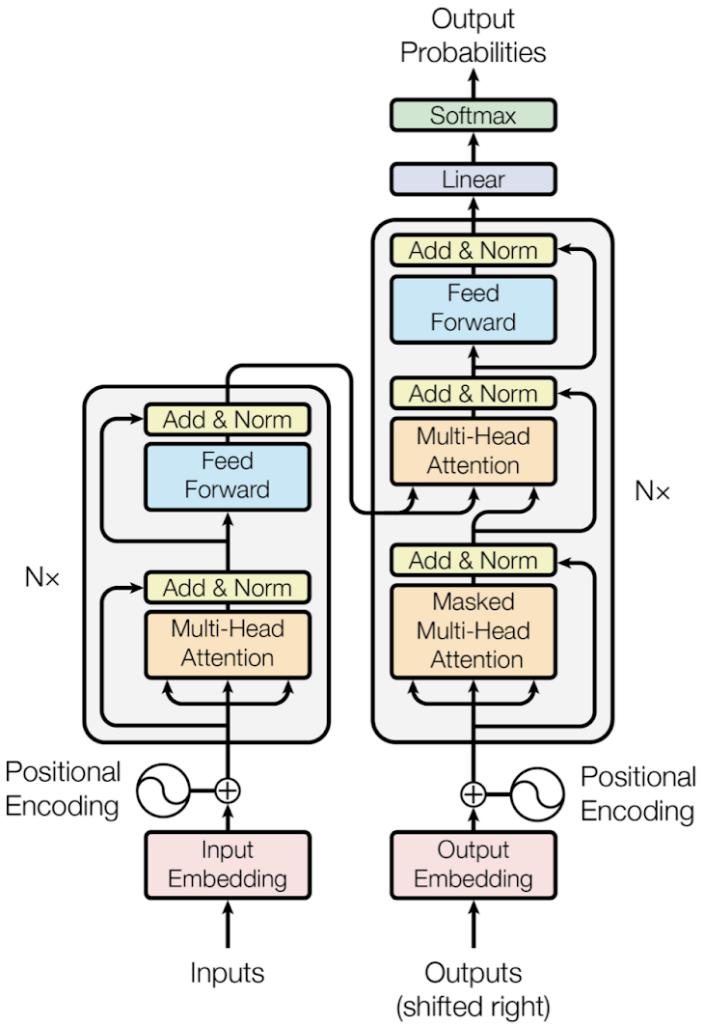


Figure 13: Architecture of a *transformer* (source: *Attention is all you need*)

- With the help of the decoder of the VAE, one then obtains a vector of size d_{model} which one gives as memory to a *transformer* similar to the first to reconstruct the encoded sequence (class `TransformerDecoderLayer` of `PyTorch`). By a mechanism of masked attention, this *transformer* successively generates vectors $\tilde{Z}_1, \tilde{Z}_2, \ldots, \tilde{Z}_l$, where the *masking* ensures that the generation of $\tilde{Z}_i$ is the result of the attention on $\tilde{Z}_1, \ldots, \tilde{Z_{i-1}}$ only.

- Finally, each $\tilde{Z}_i$ is projected into a vector of size $N = 24$ thanks to a linear layer to give the reconstruction of the *initial frame*.

The architecture of the model is represented in figure 14.

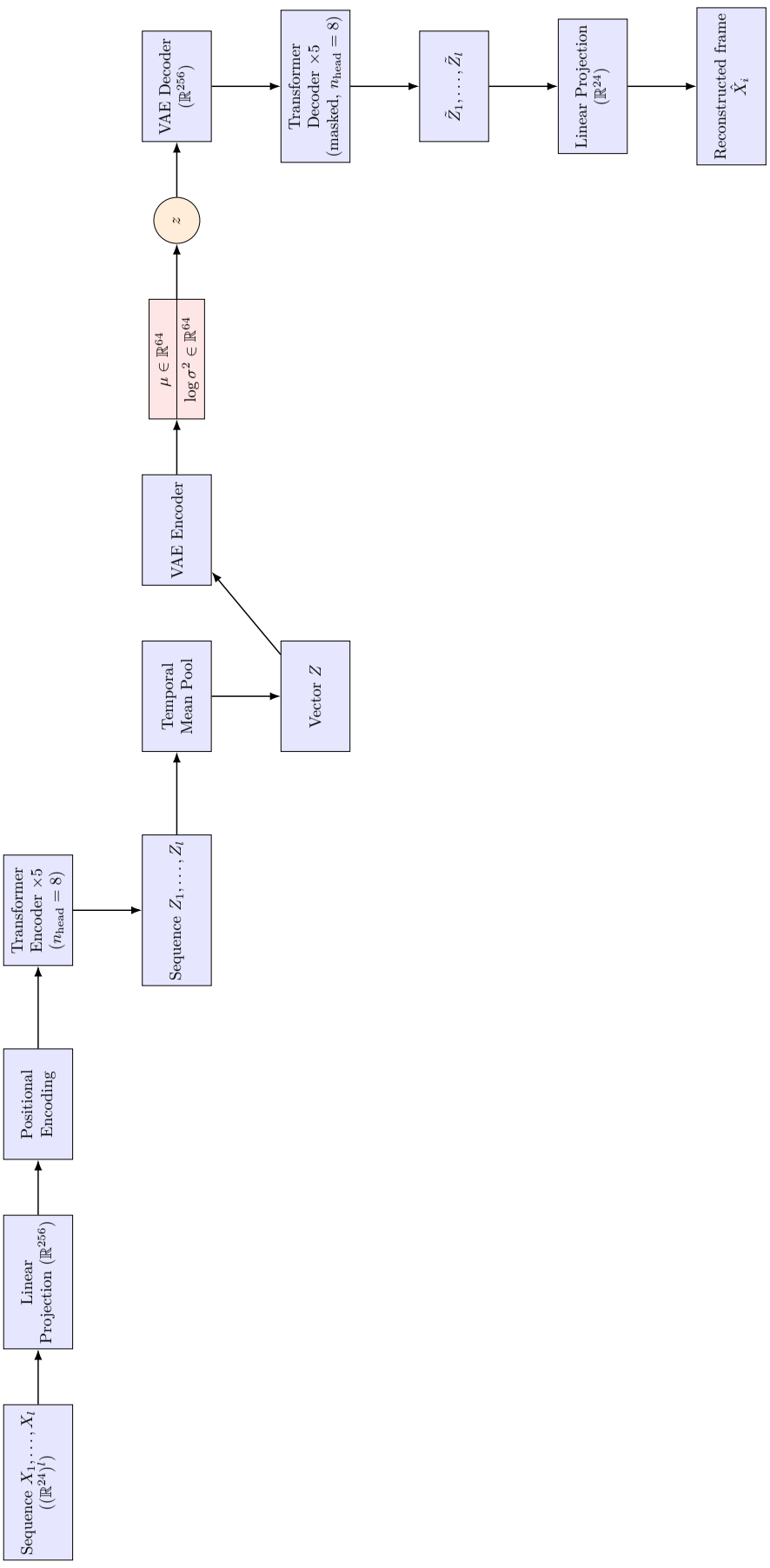


Figure 14: Complete architecture of the model: *Transformer* encoder, VAE and *Transformer* decoder with masked attention.

6.2.3 • A non-sequential model

In parallel with the sequential model mentioned above, we also studied a theoretically simpler model, non-sequential (encoding and decoding sequences of fixed length) and based on another technology, convolutional neural networks (CNN).

Proposed, formalized and implemented for the first time by Yann LeCun, accompanied by Léon Bottou, Yoshua Bengio and Patrick Haffner, in the 1980s–90s in the article *Gradient-Based Learning Applied to Document Recognition* [5], CNNs find their main application in the processing of 2D images (classification, segmentation...) since their functioning (based on convolutions and *poolings* of adjacent pixels) allows to capture the local characteristics of the images given as argument. However, it is possible to adapt these architectures to sequences of images (videos), by simply considering the temporal axis as a new axis on which the convolutions are performed.

The idea of using CNNs came to us from the article by the researcher of the Institut Pasteur Alexandre Blanc *Statistical signature of subtle behavioural changes in large-scale behavioural assays* [2], who implemented them to build a sequence encoder. Indeed, their use seems more promising than a classical MLP since these networks can capture more finely the positions of the larva.

Thus, the model that we tried to implement to generate larva trajectories of 30 *frames* also had an auto-encoder type architecture. Each *frame* was converted into a matrix of 128 pixels by 128 containing 0 and 1 (in type `numpty intu8` to save memory) and was passed through a succession of 3D convolutional layers with *stride* greater than 2 on each coordinate to "compress" progressively the sequence.

Once this sequence was encoded into a tensor of size $128 \times 8 \times 8$, this one was flattened into a vector and given to a VAE. Then, a decoder of similar architecture but using `ConvTranspose3d` layers was used to reconstruct the initial *frames*.

More precisely, the model returns a matrix of size 128×128 containing the probabilities that each pixel is white. To reconstruct the initial image, we then round these values to the nearest integer. To compute the loss resulting from passing the sequence X into the model, we use the binary cross-entropy loss per image, while weighting the KL term by a factor β. The loss used for the training of this model is therefore:

$$\mathcal{L} : (X, \hat{X}) \mapsto \frac{-1}{l} \sum_{k=1}^{l} \sum_{i=1}^{128} \sum_{j=1}^{128} [X_{i,j}^{(k)} \ln(\hat{X}_{i,j}^{(k)}) + (1 - X_{i,j}^{(k)}) \ln(1 - \hat{X}_{i,j}^{(k)})] + \beta D_{KL}(q_\theta(z|f_{\text{enc}}(X))||p(z))$$

where we denoted $(X_{i,j}^{(k)})$ the matrix obtained by converting the *frame* k of the sequence X into image, and we denoted f_{enc} the encoding function of the CNN3D (without VAE).

The architecture of the model is represented in figure 15.

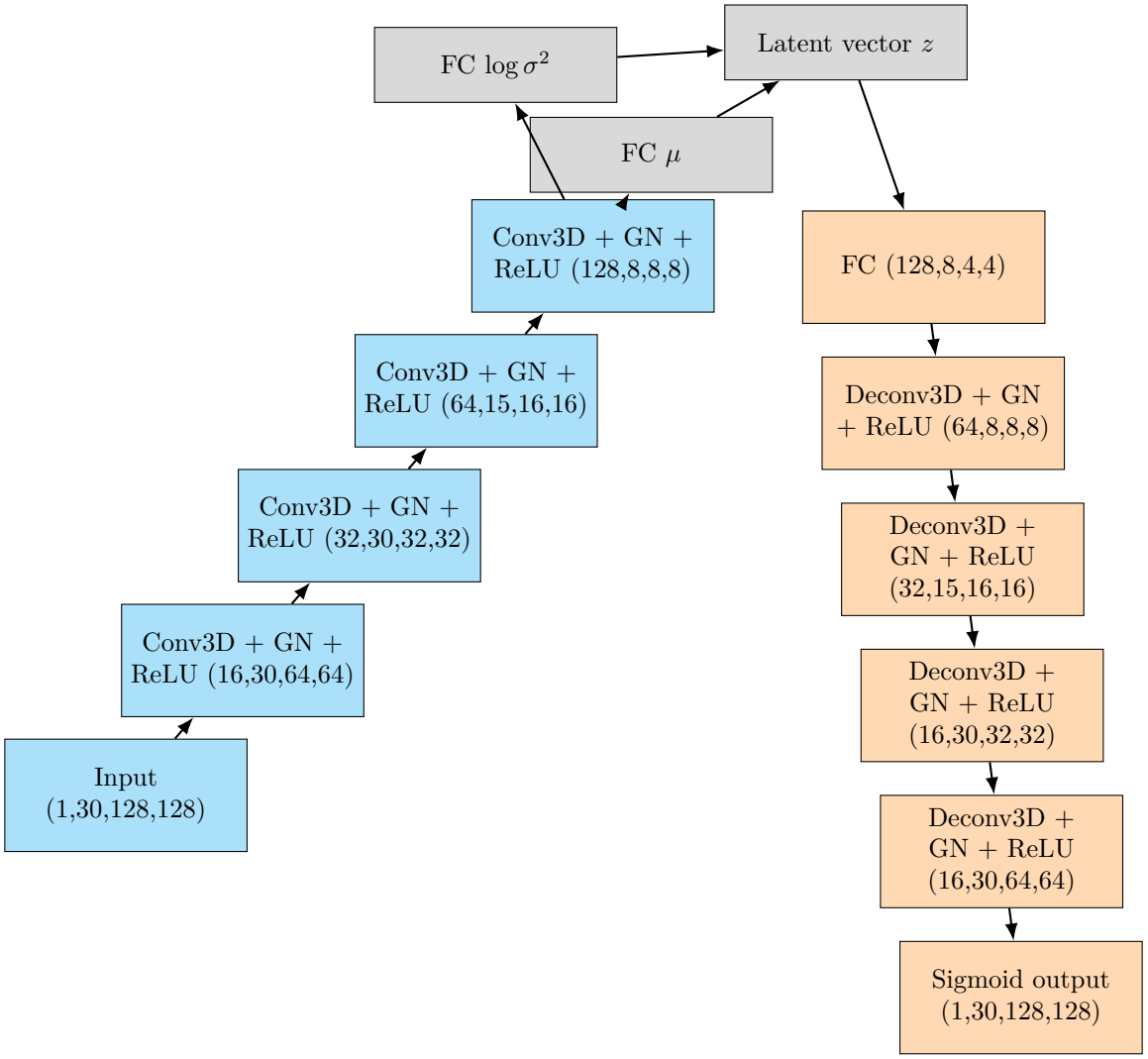


Figure 15: First architecture of the CNN3D + VAE. All convolutional and deconvolutional layers include group normalization (GN) and a ReLU activation (except final layer: sigmoid).

7
TRAINING, RESULTS AND DIFFICULTIES ENCOUNTERED

7.1 Transformer VAE

- Dataset

Having read that *transformers* are quite data-hungry, our objective was to create a dataset containing both enough sequences in total, but also enough sequences of each type of movement. To get an idea of the distribution of the different types of movement, we fixed a maximum length of 50 *frames* and indexed all the sequences of a folder of the database according to the majority type of movement they presented.
This folder being very large (more than 7000 sub-folders, one for each video, each containing the trajectories of several dozen larvae over durations that could go up to more than one minute), we resorted to *multiprocessing* thanks to the library of the same name, which allows to distribute the tasks (that is, the processing of each sub-folder) over the different processors available, thus allowing us to index all of these sub-folders in less than two hours.

More precisely, we stored in a `.h5` file for each sequence the index of the sub-folder in which the sequence is located, the index of the larva in the video, the start instant of the considered sequence, the end index, and the action carried out in this sequence.

The result is given in table 1:

Type of sequence	**Number**	**Proportion**
Total number of sequences	6 226 844	100.0%
Runs	2 988 033	48.0%
Casts	2 956 595	47.5%
Backs	157 585	2.5%
Hunchs	80 975	1.3%
Rolls	43 656	0.7%

Table 1: Distribution of sequence types in the dataset

We immediately notice the tiny proportion of *rolls*, which limits our ability to build a large dataset. In order to constitute a training dataset that resembles both the "natural" distribution of types of movement, but with a slight over-representation of the under-represented movements, we designed a training dataset whose composition is given in table 2.

We also made sure to build a validation dataset whose distribution of movements is the same as that of the training data. This one contains 20,000 sequences, that is 1% of the number of sequences of the training dataset (table 3).

Type of sequence	Number	Proportion
Total number of sequences	2 000 000	100.0%
Runs	895 000	44.75%
Casts	895 000	44.75%
Backs	100 000	5%
Hunchs	75 000	3.75%
Rolls	35 000	1.75%

Table 2: Distribution of sequence types in the training dataset

Type of sequence	Number	Proportion
Total number of sequences	20 000	100.0%
Runs	8 950	44.75%
Casts	8 950	44.75%
Backs	1 000	5%
Hunchs	750	3.75%
Rolls	350	1.75%

Table 3: Distribution of sequence types in the validation dataset

After having defined the proportions of each type of movement that we wanted in our dataset, we reused the index built previously to select the right number of **valid** sequences. If we had to resort to preliminary indexing, it is because it was not possible to select directly the desired number of sequences with `multiprocessing` since the processes taking place on each processor do not have real-time access to the number of sequences selected so far. We thus ended up with significantly too many sequences.

Thus, we grouped the indexed sequences by sub-folder and browsed part of them to identify **the right number** of valid sequences. Finally, we reused `multiprocessing` to distribute the preprocessing of the sequences (*min-max scaling*, *smoothing*) among the processors, flatten the sequences thus processed and store them in a 5GB `.h5` file for the training dataset.

- Training

The training gave us extremely hard time. Indeed, we experienced the phenomenon of *posterior collapse* which we took a long time to diagnose and to treat. *Posterior collapse* is the phenomenon whereby the network learns to ignore the latent variable z. Concretely, the encoder part assigns to each variable a law $\mathcal{N}(0, I)$: the KL divergence is then very close to 0 but the latent variable z no longer conveys any information about the initial sequence X. After training our model for 35 *epochs*, we were able to observe this phenomenon on figure 16.

Having read that this problem could be due to too strong an influence of the D_{KL} term of the loss function in the early phases of training, we first tried to introduce a coefficient β_t whose value depends on the *epoch* t, which increases gradually in order to weight the KL divergence. More precisely, we used the loss function:

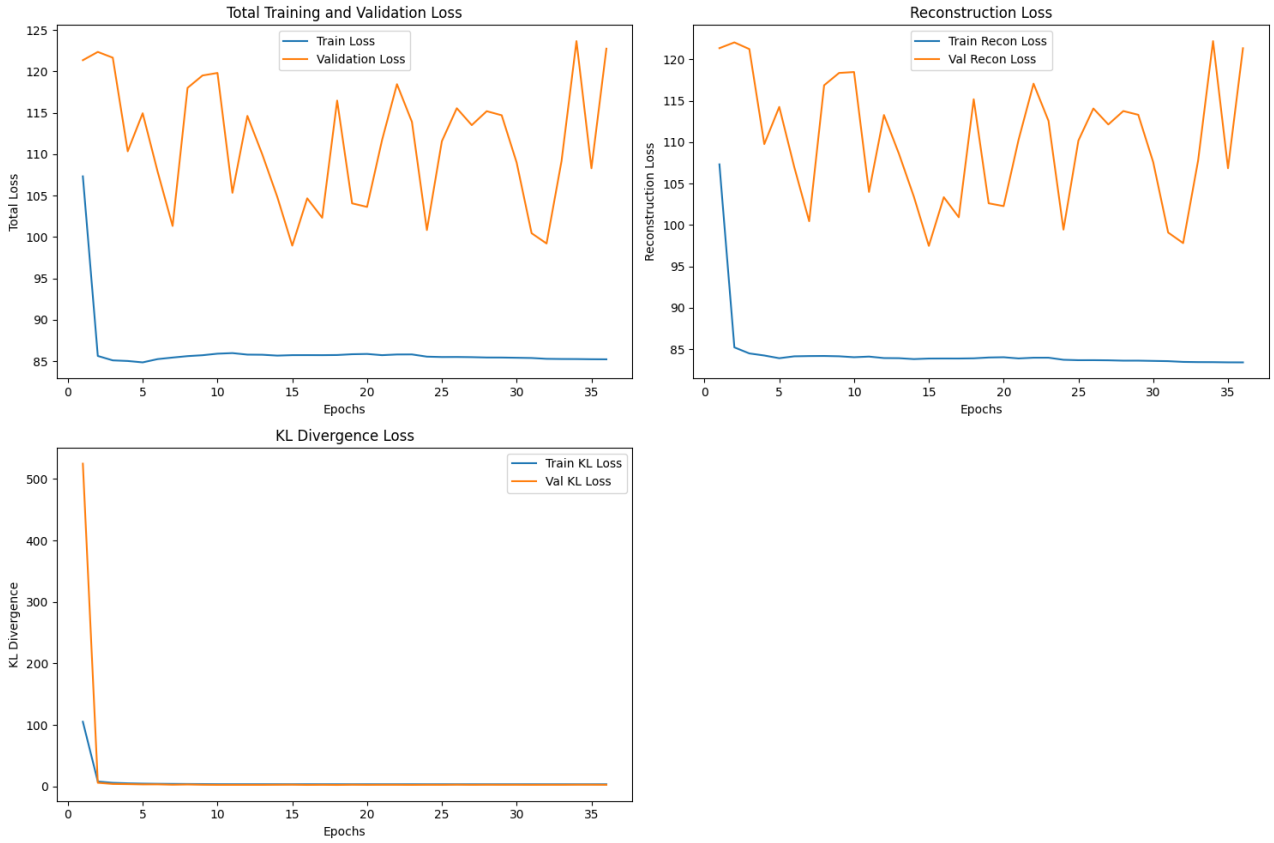


Figure 16: *Posterior collapse* phenomenon encountered during training

$$\mathcal{L} : (X, \hat{X}, t) \mapsto \frac{1}{Nl} \sum_{i=1}^{l} ||X_i - \hat{X}_i||_2^2 + \min(0.1, (t+1) \times 0.005) D_{KL}(q_\theta(z|f_{\text{LSTM}}(X))||p(z))$$

By training our model for 100 *epochs* (adding an early stopping of the training after 10 *epochs* without improvement of the validation loss after *epoch* 20, moment from which $\beta(t) = 0.1$) and choosing the optimizer `Adam` (`lr` $= 0.001$, `weight_decay`$=10^{-5}$), we obtain figure 17.

One should not be surprised by the "increasing" appearance of the total loss which is due to the variability of β.

The training being relatively time-consuming (more than 12 hours when performed on 6 graphics cards), we could not perform many attempts either. With the help of this pre-trained model, we were able to generate sequences that can be found on the Google Drive.

As we can see, the sequences are noisy, but they nevertheless present several types of movements (notably *run*, *cast* and *hunch*) which means that the model manages to capture to some extent the existence of different types of movements as well as their chaining. The noisy character of the generated sequences, for its part, could come from the input data whose noise was amplified by the model. To remedy this, we tried to add a term penalizing too great an

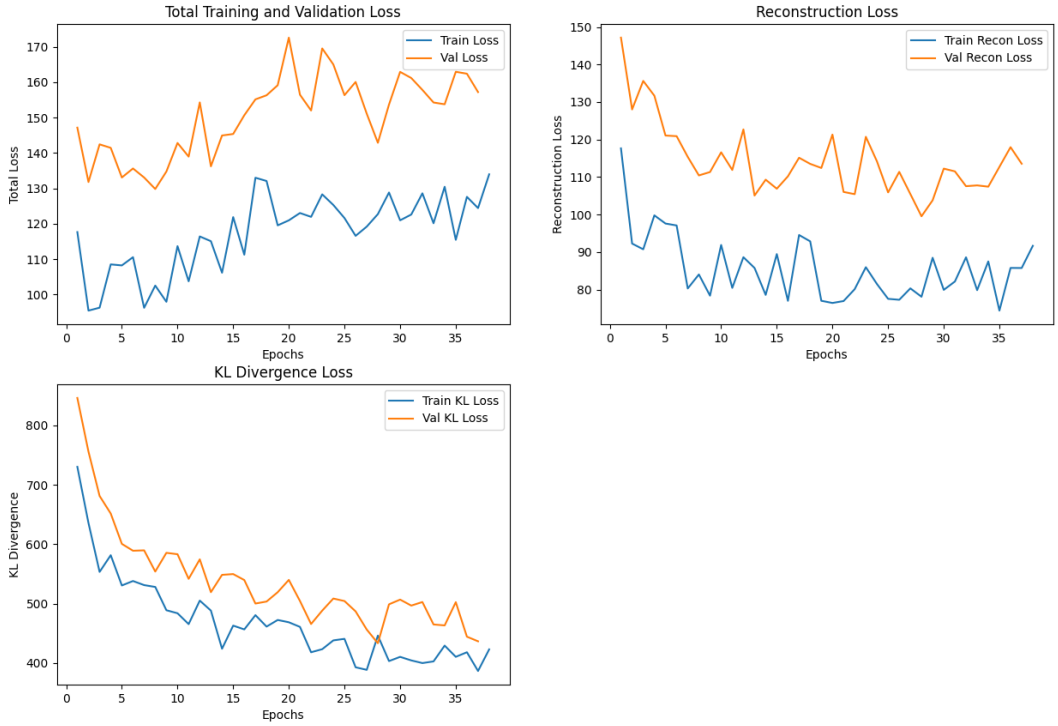


Figure 17: Evolution of the total loss, the reconstruction loss and the validation loss

acceleration in the generated trajectories, but this did not bear fruit (the generated trajectories were stationary, no matter how we weighted the penalizing term of the acceleration).

7.2 CNN3D

Let us mention briefly our attempt at training our CNN3D model, which was not at all as conclusive but which was nevertheless the pretext for some key lessons of *machine learning* techniques.

- Dataset

The dataset used for the training of this model was a sub-dataset of the previous dataset consisting of 300 000 sequences of similar distribution, but each of the *frames* was converted into an image of 0 and 1 of 128 pixels by 128 before the training, the 0 pixels being those of the contour of the larva.

- Training

The attempt at training this model was the occasion to discover several typical problems.

First of all, we observed that the training failed from the first *epoch* because of an overflow error of the maximum number representable on our system. By examining further, we noticed that the coordinates of the means μ and of the variances σ^2 of the latent space exploded, a sign that the KL term of the loss did not constrain the model enough to keep the Gaussian distributions resulting from the encoding "close" to $\mathcal{N}(0, I)$. However, after having increased the factor β weighting the KL term in the loss function, we this time encountered the opposite problem, namely *posterior collapse*.

This great sensitivity of the model to the latent space then led us to implement a *skip connection*, taking inspiration from a model of type *U-net*, introduced for the first time in the article *U-Net: Convolutional Networks for Biomedical Image Segmentation* by Olaf Ronneberger, Philipp Fischer, and Thomas Brox and very used in medical imaging. We also weighted this *skip connection* by a parameter learnable by the model which we ensured remained between 0 and 1 during training by composing it with a sigmoid at each *epoch*. The model thus obtained is represented in figure 18.

Despite these successive modifications, and despite our attempts to weight the different terms of the loss function, the model generated only aberrant sequences, or totally white images.

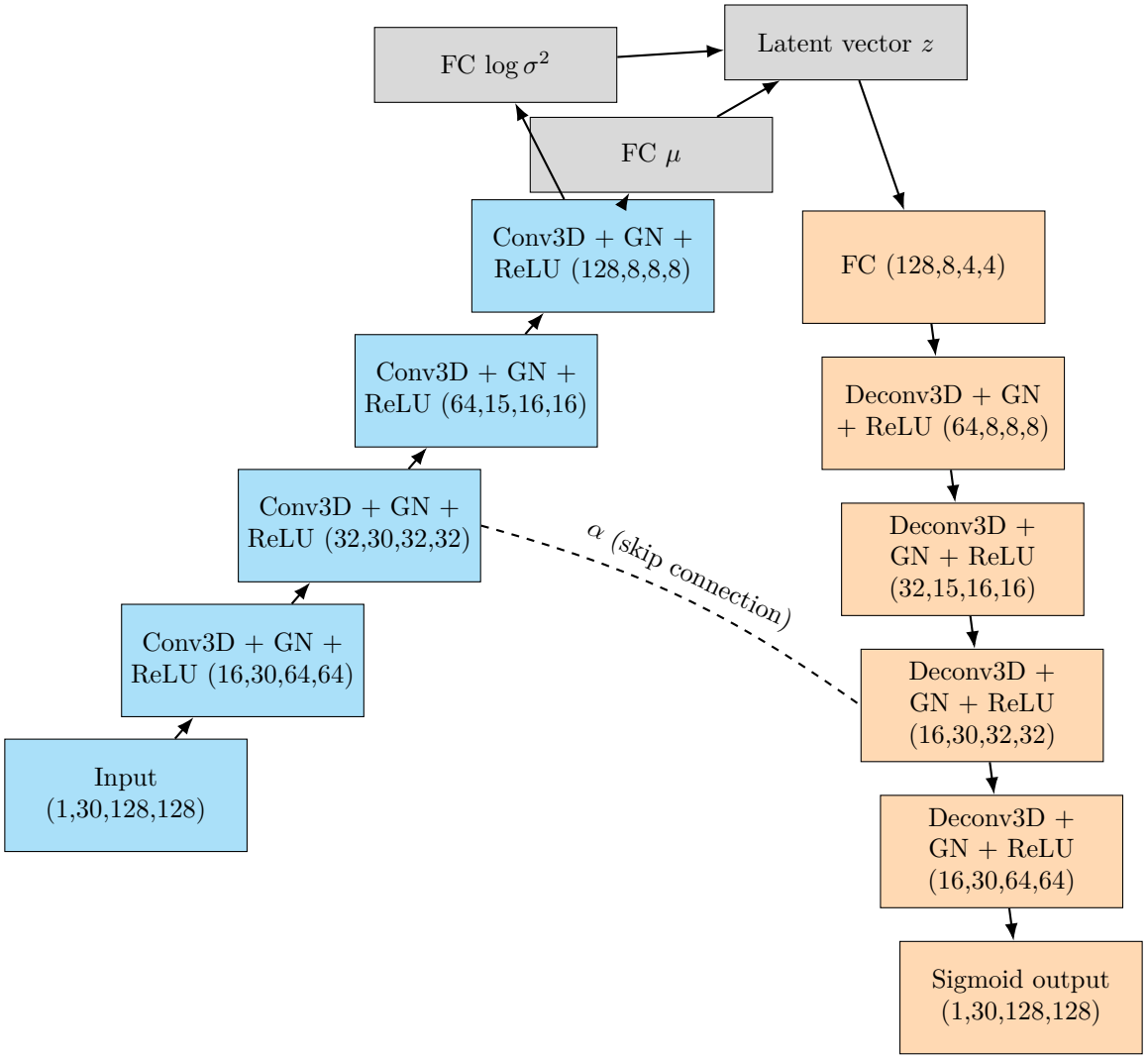


Figure 18: Architecture of the CNN3D + VAE with a weighted *skip connection* in dashed lines between the second encoder block and the second-to-last decoder block. All convolutional and deconvolutional layers include group normalization (GN) and a ReLU activation (except the final layer: sigmoid).

8
CONCLUSION AND PERSPECTIVES

This project allowed us to explore in depth the study of the influence of the genome on the behaviors of the larvae, as well as the application of artificial intelligence to the modeling of their behavior. Through the *processing* and analysis of the data, the dimensionality reduction, and the generation of trajectories via architectures such as VAEs and *Transformers*, we attempted to design a behavioral model capable of reproducing — with certain limits — realistic motor sequences.

From a theoretical point of view, this project offered us a true immersion into modern generative models, the issues of sequential learning and the challenges of latent representation. We were notably confronted with classical but tenacious problems of *machine learning* such as *posterior collapse*.

On the methodological level, we learned to work as a true research team. The rigorous use of Git and GitHub for version control, the use of an HPC cluster (*Maestro*) for the training of costly models, and the collaborative structuring of the work (distribution, *merge requests*, cross-validation of the code) deeply reinforced our skills in software engineering and in collective scientific practices.

Perspectives. This work opens many paths. But beyond generation, an exciting direction would be to analyze what the models learn: which dimensions of larval behavior are encoded in the latent space? Can some latent directions be associated with motor patterns or genetic effects? Do the attention vectors in the *transformers* reveal biologically plausible dependencies? A systematic study of the internal behavior of the model could offer not only interpretability tools, but also new neuromechanical hypotheses, to be confronted with experiment.

This project thus marks a first step in our path as researchers. It allowed us to pass from the use of models to their design, from the reading of articles to the writing of code, and above all, from the understanding of the tools to the scientific intuition of their possible uses.

REFERENCES

[1] Masson, J.-B., Laurent, F., Cardona, A., Barré, C., Skatchkovsky, N., & Zlatic, M. (2020). Identifying neural substrates of competitive interactions and sequence transitions during mechanosensory responses in Drosophila. *PLOS Genetics*, **16**(2), e1008589.

[2] Blanc, A., Laurent, F., Barbier–Chebbah, A., Cocanougher, B. T., Jones, B. M. W., Hague, P., Chikhi, R., Vestergaard, C. L., Jovanic, T., Masson, J.-B., & Barré, C. (2024). Statistical signature of subtle behavioural changes in large-scale behavioural assays. *bioRxiv*. https://doi.org/10.1101/2024.05.03.591825

[3] Kingma, D. P., & Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*.

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, **30**.

[5] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

[6] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)* (pp. 234–241). Springer.