AML 23/24 - Assignment 3

Custom model, final report

Giacomo Bellini, bellini.1970896@studenti.uniroma1.it
Ludovica Mazza, mazza.1917778@studenti.uniroma1.it
Matteo Rampolla, rampolla.1762214@studenti.uniroma1.it
Daniele Solombrino, solombrino.1743111@studenti.uniroma1.it

Introduction

This is the final report regarding the “Your Custom Model” section of the third assignment for AML 23/24.

This report illustrates the entire **process** that helped us **develop** our **custom** models, including our thoughts and observations, fuelled by the theoretical AML lectures.

We experimented with **four** different **setups**, namely `ConvSTGAT`, `STACFormer`, `STCFormer` and `STCFormer-autoregressive`, and achieved train, validation and test **A-MPJPE losses < 80 mm** in `STCFormer`.

These experiments allowed us to practice **literature** research, adapting open-source code to our own needs and hyper-parameter fine-tuning.

STCFormer

After studying some open-source Transformer implementations, like this one, we looked for spatio-temporal attention papers and/or codes.
We came across GAST-Net (paper, code), MixSTE (paper, code), MotionBERT (paper, code), PoseFormer (paper, code) and STGAT (paper, code), but all of them included some non-trivial components in their pipelines and we wanted something easier to deal with, considering it was our very first practical exposure to Transformers, outside of theory seen in class.

Finally, we found STCFormer (paper, code), which uses a very simple, yet effective, trick.
After mapping the input features to an arbitrary hidden dimensionality and applying the query, key and value transformations, they **divide** the results into two chunks, in order to apply **two different attentions**: one along the **temporal** dimension and one along the **spatial** dimension.
They argue that this creates a “Criss-Cross Attention”, which increases the performance of 3D human pose estimation tasks.

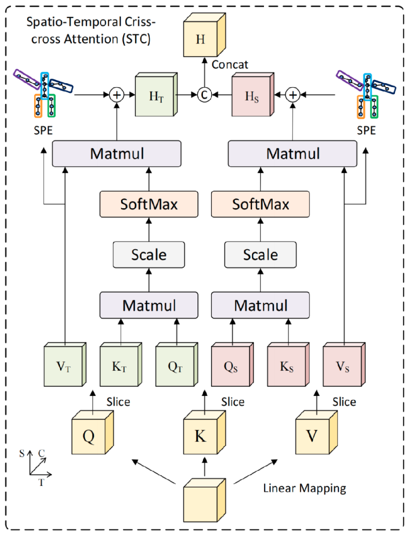

Figure 1. Spatio-temporal cross-attention schematics from STCFormer paper

Additionally, this design easily allows the use of the same code for the self- and cross-attention in the Transformer, since it simply becomes a matter of what inputs are sent in as query, key and value.

STCFormer itself is not a Transformer, so we took their Criss-Cross Attention and **implemented** it in a Transformer architecture, by means of including it as a layer in our Encoder and Decoder Blocks.

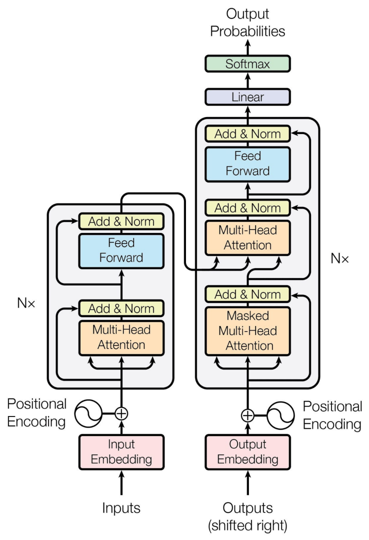

Figure 2. Transformer architecture blueprint

The Encoder takes as input `num_frames` frames (usually 10) and outputs a hidden/latent representation of them (the memory), while the Decoder processes as input `num_frames_out` frames (usually 25).

This led to an **issue** related to the **temporal dimension**, since cross-attention requires queries, keys and values to have the same temporal size which is not guaranteed in our case.
We decided to use a 2-dimensional convolutional layer to “explode” the temporal dimension of the query in the cross-attention code, mapping `num_frames` input channels to `num_frames_out` output channels.
This “trick” is similar to what STTFormer does, only we decided to use a 2D convolution to have the possibility of applying some filters to the spatial dimensionality as well.
We experimented with **other approaches**, like keeping the two different temporal sizes, but the convolution-based time explosion gave better performances.

Encoder and Decoder blocks follow the same blueprint of figure 3, which is taken from the STCFormer paper

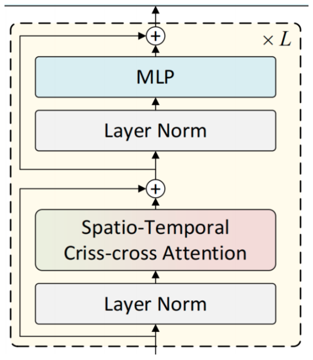

Figure 3. Encoder and Decoder blocks, as per STCFormer paper

We decided to apply **triangular masking** only to the **Decoder** attention, allowing the encoder to look at the future.
This made sense to us, considering the prototypical use-case for such architecture, in which the input `num_frames` frames are always given and needed to make a prediction.

We started with the same hyperparameters as `STTFormer`.
After hyperparameter tuning (Appendix A) we:

- Managed to get **A-MPJPE** train, validation and test losses of 83.33, **71.88** and **77.7 mm**.
- Discovered a huge benefit from **bigger** query, key and values **transformations**, as well as larger matrices for the input features→hidden features→output mappings.
- Found **quasi-invariance** of losses w.r.t. number of attention heads.

The last point appeared very interesting to us.

We speculate that if using multiple heads we get the same results, then all these heads capture either the same patterns or different patterns with almost the same expressive power.

For this reason, we decided to give a look at the **attention heads** of the trained model, across the temporal (figure 4) and spatial (figure 5) dimensions both.

We did this by loading the checkpoint with best validation loss, taking its query and key performing matrix multiplication on them.

In order to understand whether the temporal and spatial attentions attend to the same or similar elements, we computed their **distance**, applying an L_1 distance formula to the two attention matrices.
Attention across the two dimensions differ by one order of magnitude distances, which would suggest that the **spatial** and **temporal patterns** discovered by the spatio-temporal attention are **similar**.

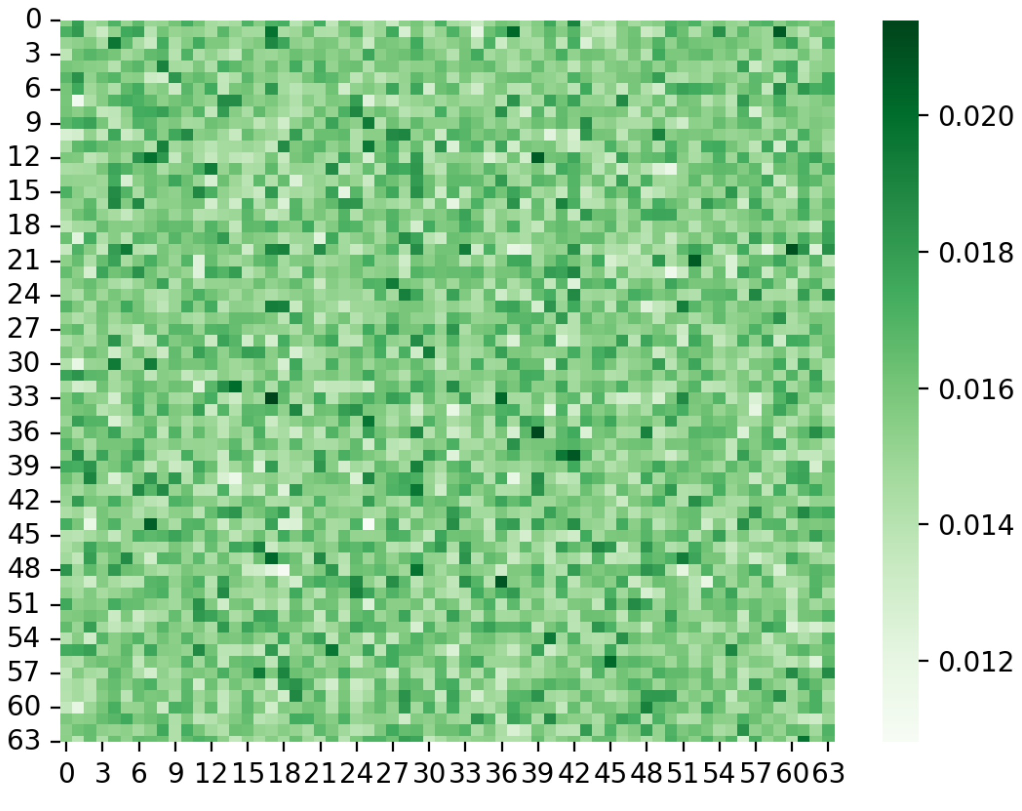

Figure 4. Softmaxed temporal attention scores, encoder block 0 (for the temporal dimension)

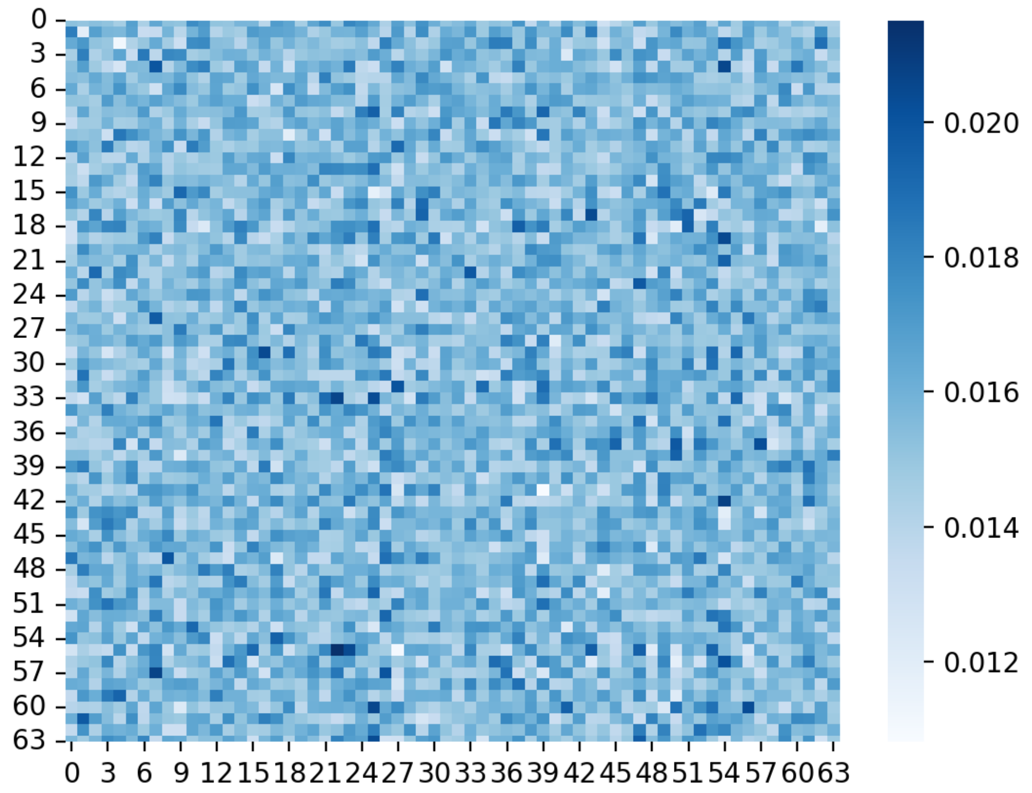
Figure 5. Softmaxed spatial attention scores, first encoder block

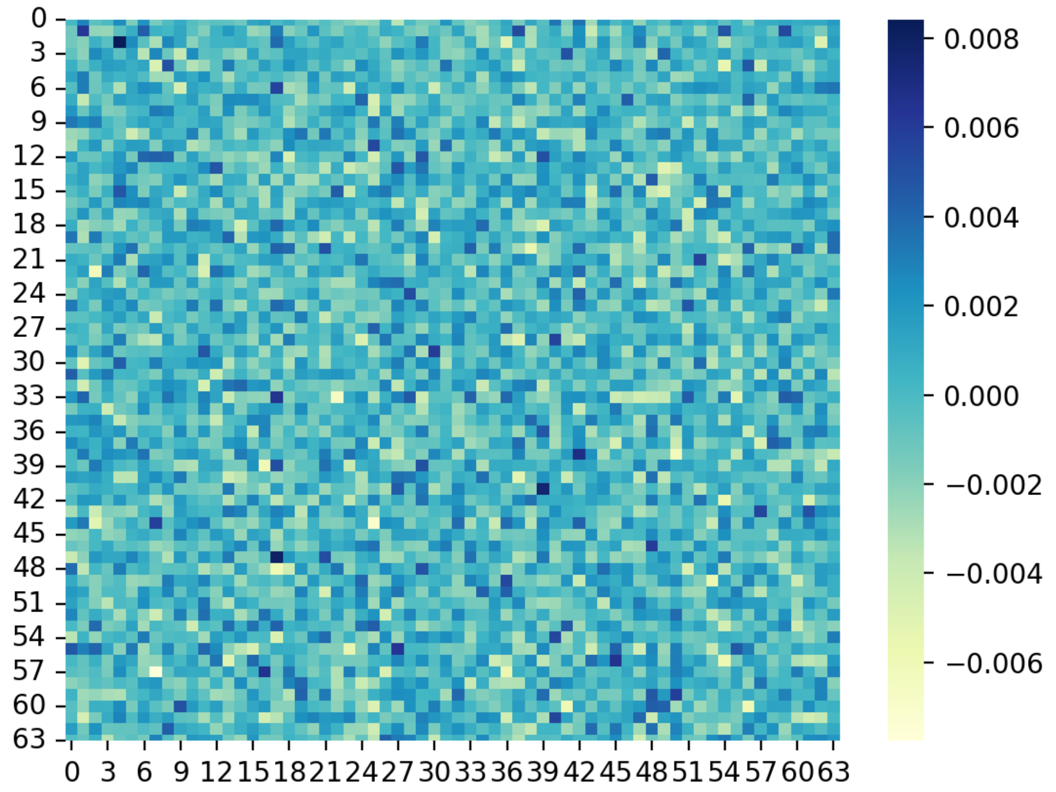
Figure 6: Distance between spatial and temporal attention scores, first encoder block.

STCFormer-autoregressive

Looking back at the current `STCFormer` setup, we realized that it is not really depticing the actual use-case of such a model, for when it will go into production.
In fact, when deployed, the model will have access to just the `num_frames` input frames (10, for example) but it will not have access to the `num_frames_out` output frames (25, for example). Rather, it should start predicting, according to the `num_frames` frames seen up until now. Basically, this means that we need to validate and test our model using the autoregressive regime.

Everything presented in STCFormer is still valid, since it can be seen as the **pre-training step** of a hypothetical “pre-train, then fine-tune” pipeline, similar to what currently happens in the NLP domain.
This makes a lot of sense considering the existence of very well known and adopted 3D human pose estimation datasets in the community and there are a lot of pre-trained models on those datasets.
For example, one may take the model pre-trained on one dataset and fine-tune it on another downstream, pose-related task, using a specific dataset for the aforementioned downstream task.

Implementation details

In order to implement STCFormer-autoregression, we **modified** STCFormer train and inference code in order to accommodate a **new loss** and perform the **autoregressive predictions**.

Differently from typical NLP autoregression, we do not necessarily need to use the end-of-sequence special token.
In fact, in language modeling, stopping the autoregressive prediction after a fixed number of outputs may yield to incomplete, meaningless sentences, while in our case it perfectly fits the ground truths in our dataset.

Taking inspiration from this paper, we encoded the start-of-sequence token/signal as the 2-dimensional one-hot vector `[0, 1]` and appended its content to the three features of our dataset.
We appended the `[0, 0]` vector to the additional frames and, had we added support for the end-of-sequence-token, we would have assigned it the `[1, 0]` vector.

Differently from `STCFormer`, `STCFormer-autoregressive` has to take into consideration these special tokens in its loss too, so we dedicated a specific loss term to them, the L_2 distance between the predicted special token and its actual ground truth value.

We could have included this new distance in the already-in-use A-MPJPE loss, but we decided to separate them considering the very **different numerical scales** of the feature and special token vectors.
Additionally, having two separate terms allows for more **advanced loss configurations** (e.g. different weights for the two terms) and analytics.

The final loss formulation is: `loss = mpjpe_error(pred, gt) + l_2_error(special_tokens_pred, special_tokens_gt)`

Similarly to `STCFormer`, we dedicated quite some time to handle the different temporal sizes across encoding and decoding.
After experimenting with **temporal masking**, which consisted in zero-padding future temporal dimensions, in order to signal to the model their absence, we resorted to the same convolution-based temporal "explosion" of `STCFormer`, which gave us better validation and test performances.

We started with the same hyperparameters as `STTFormer`.
We repeated hyperparameter tuning from scratch, in order to solidify our knowledge on the process (Appendix B), and we managed to achieve 112, 105 and 110 train, validation and test losses.

We tried to look at the two loss components on their own and we discovered that the model is very good at predicting whether a frame is a starting frame or not, achieving a very low L_2 special token loss.
Of course, since each entry in the dataset has just one frame labeled as start of sequence, we are exposed to some **class imbalancement**, so a **possible future** work may involve trying to mitigate this issue.

Validating `STCFormer-autoregressive` made us well aware of why a lot of papers, across different domains, like music, anomaly detection and video-processing, perform autoregression in the latent spaces ("**latent autoregression**").
Autoregression made our validation step 6x slower, compared to `STCFormer`, keeping the same dimensions for data and the batch.
We even tried to compensate for this by making the process multi-threaded, where each thread handled its own batch, reducing the slowness down to 4x.
We did not want to change the validation batch size in order to keep it comparable with `STCFormer` experiments

Conclusions

We experimented with **four** different **setups**, namely `ConvSTGAT`, `STAFormer`, `STCFormer` and `STCFormer-autoregressive`, and achieved train, validation and test **A-MPJPE losses < 80 mm** in `STCFormer`.

STCFormer is a **spatio-temporal Transformer** whose Encoder and Decoder use the spatio-temporal “**CrissCross**” **attention** described in the STCFormer paper, while STCFormer-autoregression is STCFormer with validation and test performed in an **autoregressive** way.
We adapted our autoregression implementation and loss from this paper.
STCFormer achieves 83.33, 71.88 and 77.7 mm of train, validation and test A-MPJPE loss, while STFFormer-autoregressive reaches 112, 105 and 110 train, validation and test losses.

We gave a look at the spatial and temporal attentions learnt by STCFormer, which made us discover some level of **similarity** between **spatial** and **temporal patterns** discovered by the model itself.

All the experiments allowed us to practice **literature** research, adapting open-source code to our own needs, hyperparameter fine-tuning, hypothesis formulation and testing.

Appendix A - STCFormer hyperparameters tuning

In this appendix, we show the entire process used to perform hyperparameter tuning of the STCFormer experiment.

For each notable configuration, we link its relative Weights and Biases page (using the run name assigned by WandB), where all details can be found.
All loss plots have the epoch on the x axis and the loss on the y axis.
Unfortunately, WandB doesn't allow to label the vertical axis.

We kicked-off using these settings (classic-moon-16):

- Batch size → 256
- Number of batches → train 71, validation 12 (10%)
- Encoder and decoder → 2 blocks, 1 attention head
- Hidden features → 32
- Total number of trainable parameters → ~23k
- Learning rate → 1e-4
- Learning rate scheduler → none
- Optimizer → Adam, with AMSGrad enabled
- Weight decay → 1e-5
- Number of epochs → 50

We immediately observed the **absence** of **overfitting**, since train (**blue**) and validation (**green**) losses go down both.

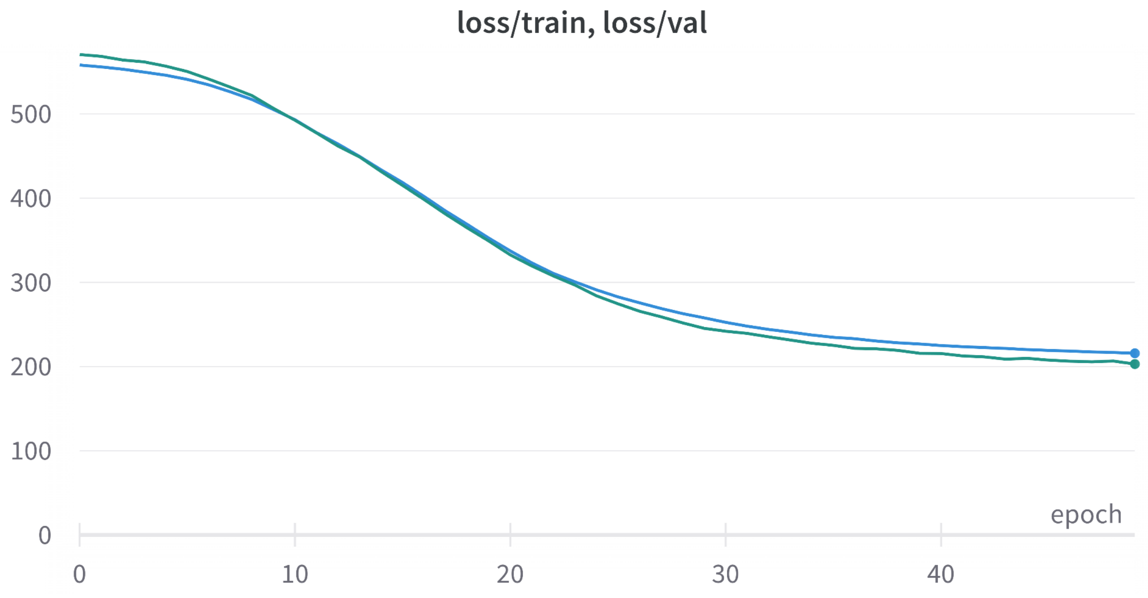

Figure A1: **train** and **validation** loss curves for classic-moon-16

Since the loss curves slowly decrease, we tried one order of magnitude faster **learning rates** in earnest-lion-17 (1e-3), polished-deluge-18 (2e-3), deft-vertex-19 (4e-3), resilient-fire-20 (8e-3) and earthy-snowflake-23 (1e-2).

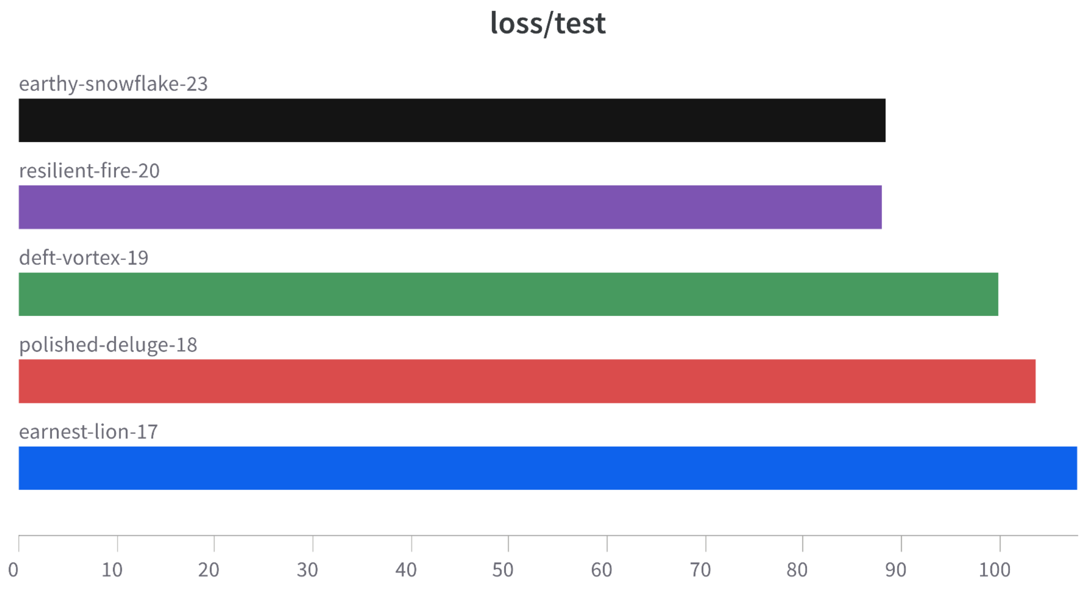

Figure A2: average test loss for runs 17, 18, 19, 20 and 23

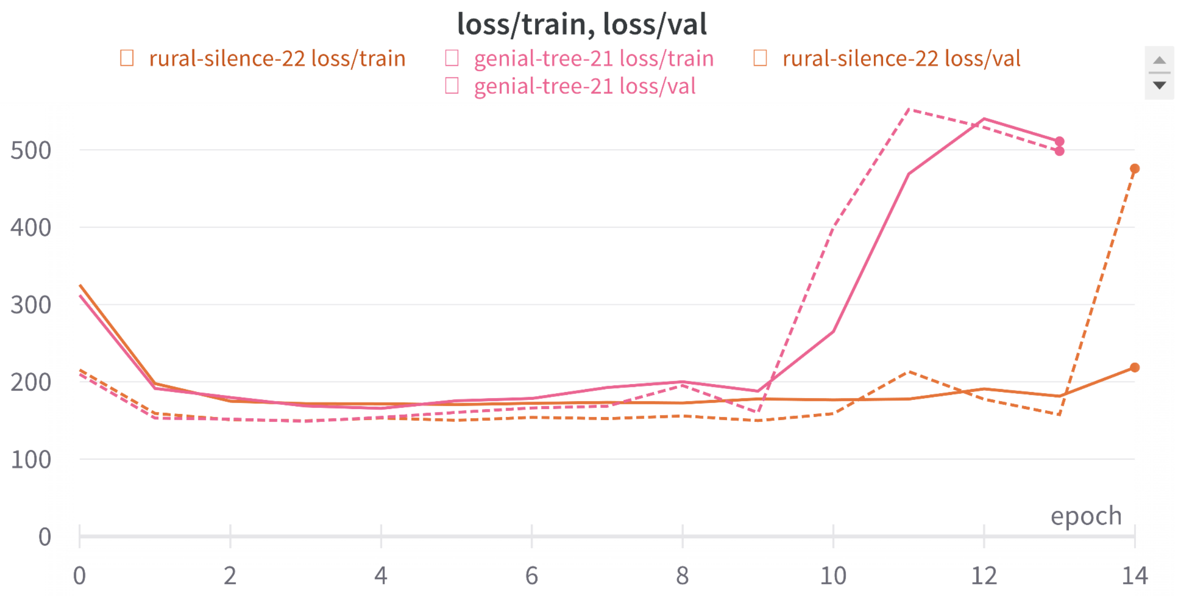

Figure A3: faster learning rates in 22 and in 21 make model overshoot

20 results in the overall smaller train, validation and test A-MPJPE losses (90.304, 77.97 and 87.9 mm)
Faster LRs (1.2e-2 in rural-silence-22 and 1.6e-2 in genial-tree-21) cause **divergence**.

Before moving on, we decided to validate results in 20 using more data.
We tried 20% batches in eager-dawn-25, 30% in gentle-plant-24, 50% in fine-snowball-26 and 100% in icon-sponge-27.
27 confirms our findings, resulting in 83.76, 71.13 and 79.8 mm of train, validation and test A-MPJPE losses.
Just to be sure, we repeated 27 and got very similar results in glowing-snowball-28.

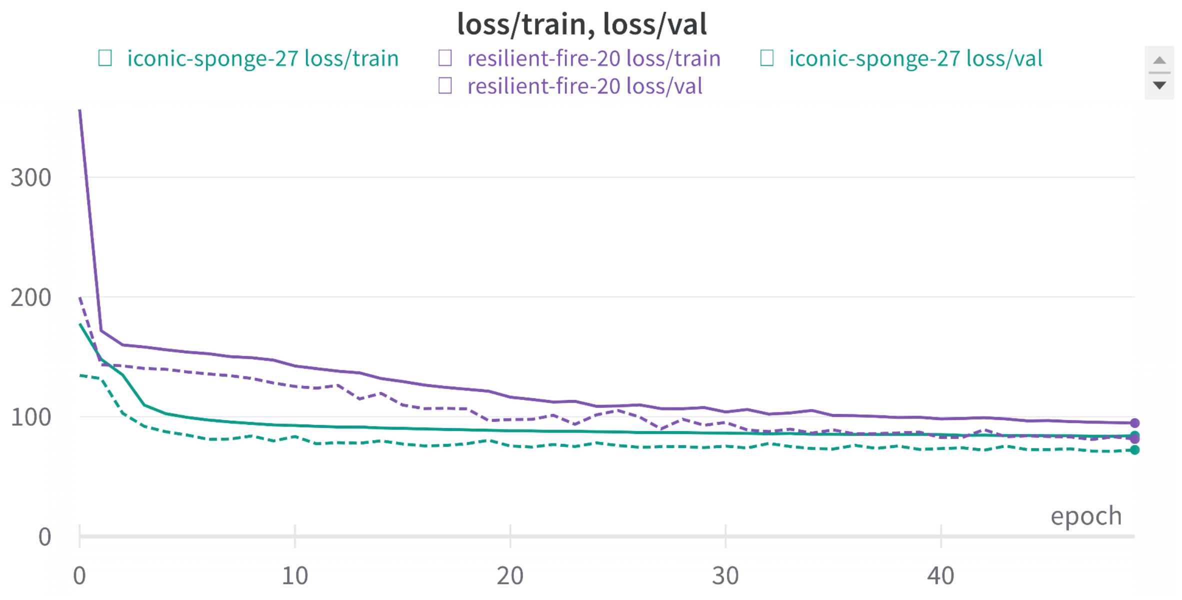

Figure A4: 27 (100% batches) losses confirm loss behavior of 20 (10% batches)

Taking 24 as reference, we decided to test whether **scaling** the **architecture up** yields performance improvements.
We tried:

1. **2x** encoding and decoding **blocks** (45k trainable parameters) in smooth-plasma-29,
2. **2x** the hidden **features** (86k trainable parameters) in comfy-cosmos-32, hopeful-microwave-35 and glamorous-hill-36.
   Note: 32 required scaling the learning rate down respectively to 2e-3, so we decided to test 35 with learning rate of 1e-3 and 36 using 8e-4.
3. **2x** attention **heads** in bright-forest-37 (without increasing hidden feature dimensionality)
4. **4x** attention **heads** in sunny-snowflake-38 (without increasing hidden feature dimensionality)

5. **2x** attention **heads** and **2x** hidden **features** in crisp-sky-40 (had to scale LR down to 2e-3 to avoid divergence)

Only one experiment in 2. (32) decreased train and validation losses by 2 mm and test loss by 4 points.

2x training parameters (23k in 24 vs. 45k in 32) for such small improvements may not be worth that much, as far as going in production is concerned.
On the other hand, from a scientific point of view, this sends us a clear message: the **model benefits** a lot from **bigger** linear **transformations** applied in self/cross-attention and {En,De}coder blocks, since increasing hidden features makes their matrices bigger.
Adding more attention heads doesn't help the model, signaling that the model doesn't need to spot a lot of different patterns along both the temporal and spatial dimension.

Since 32 is our new best performer, we decided to validate it using 50% of the batches in winter-durian-41 and 100% in faithful-glade-43 (had to scale LR down to 8e-4 to avoid divergence), which both confirm 32 results.

In STCFormer-autoregressive, we experimented with different forms of **regularization**, including dropout and weight decay.

These other regularization techniques were taken into consideration:
- Positional Encodings
- Skip connections
- Random restarts or Weights re-initialization
- DropDim
- Scaling query-key dot product

In order to move to STCFormer-autoregressive and due to time and resource constraints, we decided to set them aside for **future experiments**.

Appendix B - STCFormer-autoregressive hyperparameters tuning

Similarly to `STCFormer`, in this appendix we show the entire process used to fine-tune STCFormer-autoregressive hyperparameters.

For each notable configuration, we link its relative Weights and Biases page (using the run name assigned by WandB), where all details can be found.
All loss plots have the epoch on the x axis and the loss on the y axis.
Unfortunately, WandB doesn't allow to label the vertical axis.

We kicked-off using these settings (flowing-totem-106):

- Batch size → 256
- Number of batches → train 71, validation 12 (10%)
- Encoder and decoder → 2 blocks, 1 attention head
- Hidden features → 32
- Total number of trainable parameters → ~23k
- Learning rate → 1e-4
- Learning rate scheduler → none
- Optimizer → Adam, with AMSGrad enabled
- Weight decay → 1e-5
- Number of epochs → 40

Train and validation losses go down both, so **no overfitting** is happening.

Since the loss curves decrease slowly, we tried one order of magnitude faster **learning rates** in chocolate-pine-107 (1e-3), celestial-shadow-108 (2e-3), absurd-grass-109 (4e-3).
They all lead pretty much to the same train, validation and test losses of 113, 113 and 117 mm.
We validated 107 using 25 and 100% of batches in giddy-frost-110 and scarlet-fire-111.

110 and 111 loss curves reach a plateau (figure B1), hinting at the fact that the current configuration has expressed its maximum potential.

For this reason, we bumped the number of trainable parameters up to 86k, by doubling the hidden features to 64 in kind-silence-112.
This brought all three losses down to 112, 105 and 110 mm (from 113, 113 and 117 mm), presenting some mild signs of overfitting, starting from epoch 21.
Thanks to **early stopping**, this wouldn't necessarily be a big issue, so we decided to continue experimenting with the number of parameters.

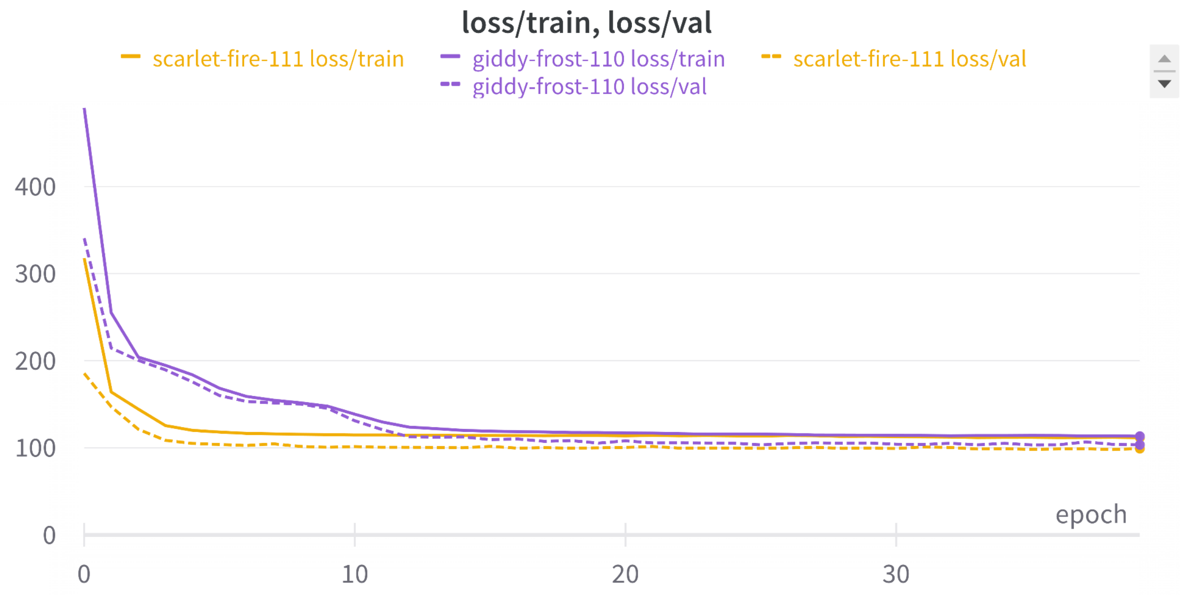

Figure B1. Train and validation losses for 110 and 111 reach a plateau after the first epochs.

Adding one block in the Encoder and Decoder in fragrant-rain-115 and dry-sunset-116 brought the number of trainable parameters up to 130k, but did not improve performances.

At this point, considering the additional time required for the slow autoregressive process and the time constraints, we decided to validate kind-silence-112 using 25 and 100% training batches in quiet-tree-152 and generous-frog-153, which validated results in 112 (figure B2 and B3)

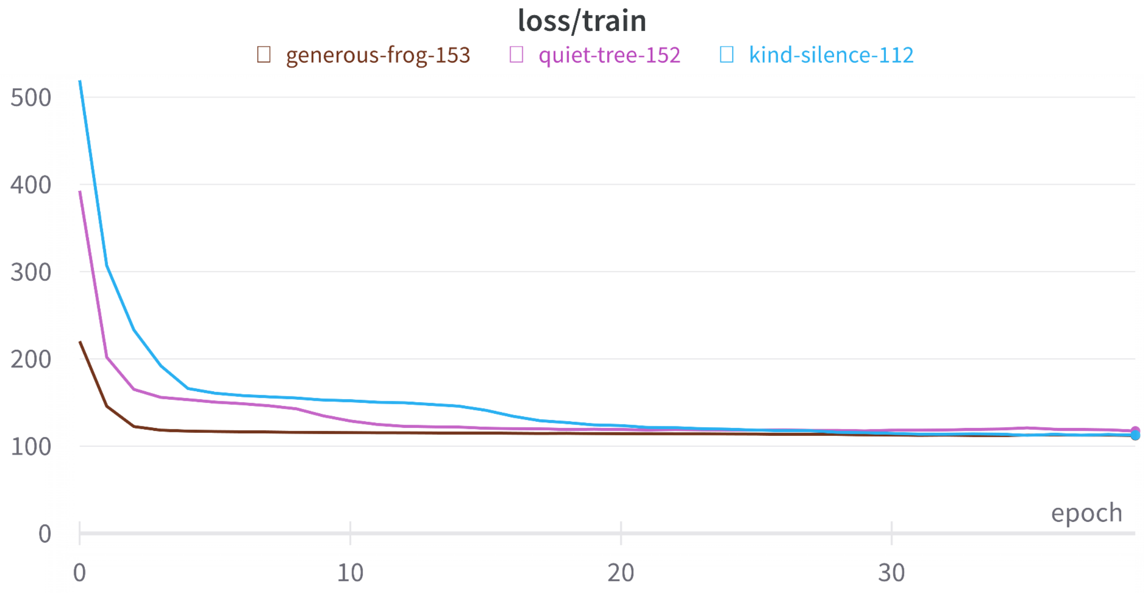

Figure B2. Train loss curves of 152 and 153 validate results in 112

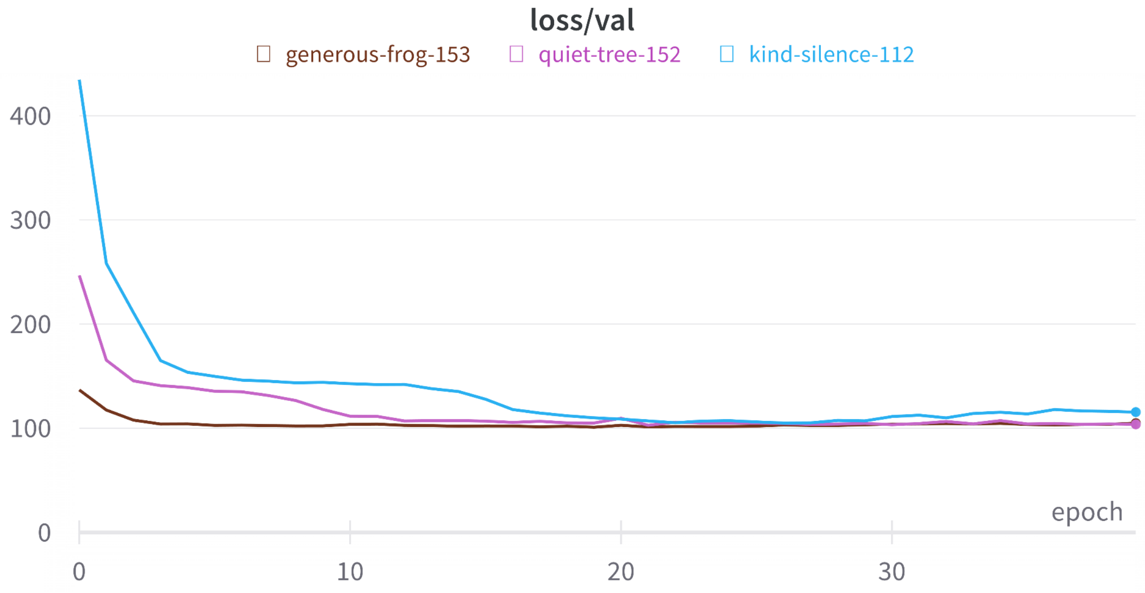

Figure B3. Validation loss curves of 152 and 153 validate results in 112

Appendix C

Before working on `STCFormer` and `STCFormer-autoregressive`, we experimented with `ConvSTGAT` and `STACFormer` and we will briefly touch on them in this appendix.

ConvSTGAT

Looking at the provided STTFormer, we noticed that the temporal expansion from `num_frames` to `num_frames_out` is entirely performed by one final convolution layer, by setting the input channels to num_frames and the output channels to `num_frames_out`.

We wanted to test whether adding more convolutional layers aiming at processing the temporal explosion would make any difference and we also tried to boost spatial performances by using graph attention layers.

We devised a ConvSTGAT block, composed of one graph attention layer and one 2-dimensional convolutional layer, and stacked several of them together in order to create a deep architecture.

In hyperparameter tuning, we also experimented with different network depths and layer disposition in the block, but performances were underwhelming, reaching 140 mm of A-MPJPE validation loss, way worse than the provided `STTFormer`, which we managed to push to XX loss in the hyperparameter fine-tuning section of the homework.

STACFormer

Since we had to eventually provide a Transformer-like architecture, we decided to start by using STTFormer as encoder and come up with our own decoder, made of STC blocks, taken from STCFormer paper which then inspired our own `STCFormer` and `STCFormer-autoregressive` solutions.

This experiment allowed us to discover that the architecture is heavily dependent on the decoder.
Once we found the best hyperparameters, we decided to fix the number of decoder blocks and to progressively reduce the number of encoder `STABlock` blocks and we noticed that using as low as just one `STABlock` keeps basically the same performances as using the default `STTFormer` number of blocks, resulting obviously in less trainable parameters and, consequently, faster training and inference.

Nevertheless, STACFormer reached 138 mm of validation A-MPJPE loss, which is on par with ConvSTGAT.