Deep Learning in Audio. Homework 4: Neural Vocoder

Alexey Slizkov
04.12.2023

This is a description of the work I have done, implemented classes and functions, and experiments.

Credits to architectures illustrations to the original paper (https://arxiv.org/pdf/2010.05646.pdf)

1 The `LJSpeechDataset` Class

The `LJSpeechDataset` class serves as a concrete subclass of `torch.utils.data.Dataset`, specifically tailored for handling the LJSpeech dataset within a neural vocoder context, in our case, the HiFiGAN vocoder. This class is integral to managing the data pipeline, from loading the audio files to transforming them into a suitable format for the neural network to consume. Below, we detail the various components and functionality of the `LJSpeechDataset` class.

1.1 Initialization

During instantiation, the `__init__` function performs several critical setup steps:

- It ensures the text encoder is not used—aligning with the assumption that this vocoder relies on pre-computed mel spectrograms rather than raw text input.
- Verifies the existence of the dataset within the given directory and checks whether the instantiated object will operate for training or validation.
- Constructs a list of audio file paths according to the specified portion of the dataset.
- Optionally shuffles the list of audio file paths if randomness is desired during training.
- Initializes a mel spectrogram converter, configured according to the `MelSpectrogramConfig`.

1.2 Retrieving Items

Each call to `__getitem__` retrieves an item from the dataset at a specified index:

- The audio file corresponding to the index is loaded, and its dimensionality is checked to ensure it is in the expected shape.
- If a segment size has been defined, a fixed-length segment is randomly selected from the audio. If the audio is shorter than the specified segment size, padding is applied.

- The audio is then processed into a mel spectrogram, which is what the neural network will actually receive as input.

1.3 Data Structure

The method returns a dictionary containing two key-value pairs: the computed mel spectrogram and the selected (or padded) audio segment. This data structure conveniently bundles the input and target data for the model.

1.4 Dataset Length

Lastly, the `__len__` method provides the total number of audio files available in the dataset section chosen, either for training or validation.

2 The `VocoderTestDataset` Class

Parallel to the `LJSpeechDataset` class used for training and validation, we have the `VocoderTestDataset` class geared specifically for the testing phase. This class responsibly handles the loading and preparing of the audio data for the final evaluation of the vocoder. Let us dissect the operational aspects of this class in the subsequent subsections.

2.1 Initialization

The `__init__` method of the `VocoderTestDataset` class encapsulates the essential steps for initializing the test dataset:

- It starts by validating the existence of the test dataset directory and collects the paths to all WAV files within.
- An assertion check ensures there are audio files present to avoid empty dataset scenarios.
- A significant cautionary message and a user-prompt mechanism are implemented to ensure the user is aware and consents to proceed, especially when any deviation from the expected three audio file test set is detected. This step requires conscious and repeated confirmation from the user to proceed with a dataset that has not been directly validated by the existing codebase.

2.2 Data Confirmation Process

A user interaction process involves a couple of layers of affirmation:

- After an initial warning, the user is asked to confirm their understanding and acceptance of potential consequences of proceeding with an untested dataset by inputting a specific agreement string.
- If the input does not match or if the confirmation does not come in a timely fashion—suggesting a lack of thoughtful consideration—the process aborts to safeguard against unintended usage or consequences.

2.3 Retrieving Items

Item retrieval through the `__getitem__` method involves:

- Loading the chosen audio file and verifying its dimensionality for consistency with expectations (single channel, two-dimensional array).
- Computing the mel spectrogram of the audio using the initialized spectrogram converter.

2.4 Data Structure

Similar to its counterpart used in training, the `VocoderTestDataset` returns a dictionary containing a mel spectrogram and the original audio data—both crucial components in the evaluation of the vocoder model.

2.5 Conclusion

The `VocoderTestDataset` class is meticulously tailored to manage a testing pipeline that accommodates the potential variability of test data sizes while ensuring user acknowledgment of any deviations from the anticipated test conditions, which is paramount for maintaining the integrity of the testing process.

3 The HiFiGAN Model

Within the HiFiGAN vocoder framework, various model components interlink to facilitate high-fidelity audio synthesis. One of the pivotal units in this architecture is the ResBlockV1 class, a fundamental building block designed to enhance the model's representation learning capabilities.

3.1 The `ResBlockV1` Class

The `ResBlockV1` module represents the essence of the residual block used in the HiFiGAN model.

- **Inspiration:** As a node to the official HiFiGAN source code, our implementation draws foundational principles from it, while tailored adjustments and reimaginings are fashioned where necessary.
- **Configurations:** Hyperparameters for the residual block, such as channel counts, kernel sizes, dilation rates, and activation steepness, are flexibly parameterized, empowering versatile instantiation to suit varying architectural demands.

3.2 Structure and Initialization

Upon initialization, the `ResBlockV1` class performs the following series of events:

1. **Convolutional Layers:** The class creates three pairs of convolutional layers. Each pair is constructed with:
    - Weight normalization to stabilize the learning process.

- Leaky ReLU activation for maintaining gradient flow even when non-linearities are introduced.
- Carefully calculated padding to ensure the dilation does not alter the input dimensionality drastically.

2. **Weight Initialization:** It uses a custom initialization method for the convolutional weights to foster optimal training conditions from the start.

3.3 Forward Pass

The forward pass of the `ResBlockV1` class showcases its residual nature:

1. **Residual Connections:** The input signal is successively processed through the convolutional stacks, with the original input being added to the output of each stack—typifying the residual design that encourages smoother gradient flow during backpropagation.

2. **Stack Integration:** Through the summing of stack outputs with the residual connections, the block circumvents issues of vanishing gradients, providing pathways for gradients to flow through deeper layers without weakening.

3.4 Residual Learning

The employment of such residual blocks is instrumental in constructing deep learning architectures that can learn intricate data representations without falling prey to the pitfalls of depth, like vanishing or exploding gradients.

4 Refinement of HiFiGAN Architecture

The HiFiGAN model architecture is further refined with the introduction of the `ResBlockV2` and `MultiReceptiveFieldFusion` classes, which add nuance and sophistication to the processing blocks. Each class has a distinct role in the overall model, contributing to its ability to capture a diverse range of audio features.

4.1 `ResBlockV2` Class

The `ResBlockV2` class, a variant of the previously described `ResBlockV1`, is streamlined for efficiency:

- **Configuration:** Like its counterpart, it is parameterized for channel counts, kernel sizes, and activation functions, but with fewer dilation rates.
- **Construction:** It composes two convolutional stacks with weight normalization and Leaky ReLU, but each tuned with distinct dilation values for processing.
- **Initialization:** A unifying initialization strategy ensures that the model's weights start from a standardized distribution to promote effective learning.

The class follows a similar forward pass, adding the output of each convolutional stack to the original input and then passing the result to the subsequent stack. This serial application of transformations with residual connections aids in combating diminishing gradients—a common issue in deep architectures.

4.2 `MultiReceptiveFieldFusion` Class

The `MultiReceptiveFieldFusion` (MRFF) module represents a higher-order composition of residual blocks, designed for a composite receptive field approach:

- **Versatility:** The MRFF class is designed to integrate multiple residual blocks with diverging kernel sizes and dilation patterns, promoting a blend of local and global acoustic feature representation.
- **Module List:** A list of residual blocks is dynamically composed to unify varying pathways of audio signal processing, with each block contributing its unique perspective on the data.
- **Forward Pass Integration:** During the forward pass, each block's perspective is calculated and then the results are integrated by averaging, hence fusing the multi-receptive-field information into a singular representation.

This design intends to enable the model to interpret audio signals over multiple scales simultaneously, capturing the intricate nuances of human speech that vary from quick phonetic changes to more gradual prosodic contours.

4.3 Conclusion on Low-Level Architecture Components

These classes—`ResBlockV2` and `MultiReceptiveFieldFusion`—serve as pivotal elements in crafting a highly responsive and adaptive HiFiGAN model. Their roles encapsulate the principles of modern neural network design, leveraging depth, width, and multi-scale processing for enhanced audio synthesis.

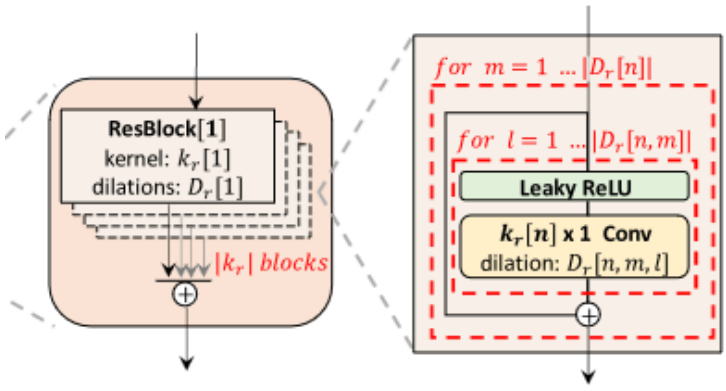


Figure 1: Illustration of the ResBlockV2 and MRFF blocks in the HiFiGAN architecture

As depicted in Figure 1, the interplay between these classes ensures that the model does not just learn patterns but also grasps the variability inherent in authentic audio signals.

5 The Generative Component of HiFiGAN

Central to the HiFiGAN vocoder is the generative network, termed the HiFiGenerator, which is adept at synthesizing high-fidelity waveforms from mel spectrograms.

5.1 Initialization of the `HiFiGenerator`

In the heart of HiFiGAN's generator lies a carefully orchestrated initialization scheme aiming to establish a foundation that is conducive to high-quality audio generation:

- **Convolutional Preprocessing:** The input mel spectrogram first encounters a preprocessing convolutional layer (`conv_pre`), designed to increase the channel dimensionality to prepare for the subsequent upsampling stages.
- **Upsampling Blocks:** The core of the generator consists of a sequence of convolutional transpose layers (upsamples). These layers incrementally upscale the representations back to the time-domain waveform's resolution.
- **Receptive Field Fusion:** Post-upsampling, the signal flows through Multi-Receptive Field Fusion blocks that blend information from various receptive fields.
- **Post-Processing to Waveform:** Finally, the `conv_post` layer caps the generator, reducing the channels to one and applying a hyperbolic tangent activation to shape the waveform properly.

Each layer and block within the generator is initialized with weights designed to foster stable gradient flow and convergence from the outset of training.

5.2 Forward Pass Dynamics

The forward pass of the HiFiGenerator incarnates the essence of generative modeling by executing the following sequence of operations:

1. **Initial Convolution:** The mel spectrogram enters the realm of the generator and is swiftly projected into a higher-dimensional space.
2. **Upsample and Fuse:** In a tandem yet orchestrated fashion, the model employs upsampling and multi-receptive field fusion techniques to enrichen the processed audio signal iteratively.
3. **Final Convolution:** As the culmination of the transformational symphony, the fine-tuned representation is converted into a raw audio waveform by the post-processing convolutional layer.

These sequential transformations enable the HiFiGenerator to breathe life into mere numerical representations, much like a maestro conducting an ensemble to produce symphonies from silence.

5.3 Conclusion

The HiFiGenerator functions as a sophisticated, generative powerhouse within the HiFiGAN vocoder, aptly showcasing the model's convolutional elegance and its capability to synthesize audio waveforms with remarkable quality.

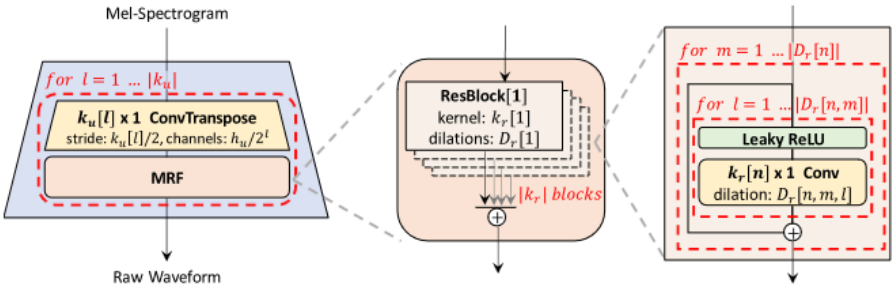


Figure 2: Diagram of the waveform generation process in the HiFiGenerator class

With the assistance of meticulously chosen hyperparameters and a thoughtful convergence of architectural elements, the HiFiGenerator stands as a testament to modern audio synthesis technology. Its role is paramount, its function elemental, and its output nothing short of extraordinary.

6 Discriminative Components of HiFiGAN

A crucial aspect of the HiFiGAN vocoder is its discriminative network, which captures and exploits the nuanced patterns within audio waveforms. This network consists of a series of discriminators, among them the HiFiDiscriminatorP and HiFiDiscriminatorMultiPeriod classes.

6.1 The `HiFiDiscriminatorP` Class

- **Design Philosophy:** The HiFiDiscriminatorP or Period Discriminator operates on the convolutional principle, inspecting audio signals with a critical eye towards periodic textures—akin to seeking rhythm in music.
- **Architecture:** It contains a chain of 2D convolutional layers with leaky ReLU activations, each designed to scrutinize the signal at varying resolutions and scales.
- **Specialized Convolution:** In an innovative twist, these layers are designed to operate across different periods of the wave, giving the model a unique insight into the periodic nature of sounds.

- **Forward Pass:** When running input data through the network, the tensor is momentarily reshaped to introduce a new dimension – a clever way to distribute the period across the batch for thorough inspection. Post that, it resumes a more traditional path through the convolutional layers.

6.2 The `HiFiDiscriminatorMultiPeriod` Class

- **Multi-Period Design:** The HiFiDiscriminatorMultiPeriod class takes the innovative concept of period-based discrimination to a new level by combining several HiFiDiscriminatorP instances, each tailored to a unique period, thus capturing a diverse spectrum of acoustic resonances.
- **Configurability:** With periods ranging from short (2) to longer (11), it is thoughtfully designed to cover a broader aspect of periodic representations within the audio.
- **Composite Features:** The composite output consists of a melding pot of features—each discriminator's view merged into a rich representation of the input data's rhythmic structure.

6.3 Implications for Audio Fidelity

The inclusion of these discriminators in the network architecture personifies the GAN's adversarial nature. Each brings a critical perspective to the generative process, ensuring that the synthesized audio does not deviate from what is perceptually authentic and true to human hearing.

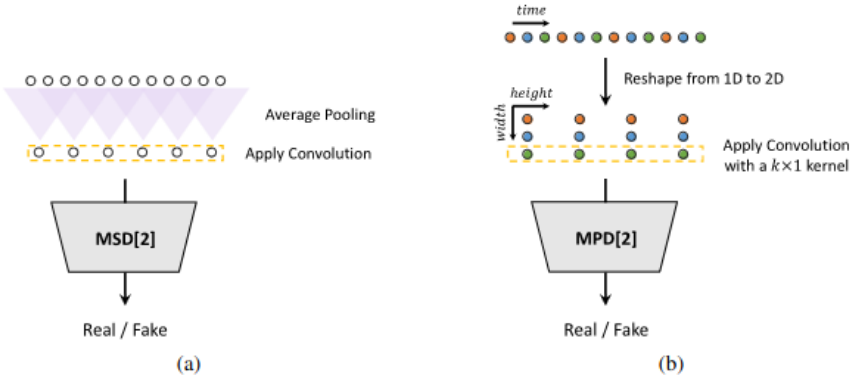


Figure 3: Schematic of the discriminating process in the Discriminators

These discriminators form a rigorous test of the generator's output, serving as a relentless jury that demands authenticity, richness, and fineness in the synthesized sound.

7 Advancing Fidelity with Scale Discriminators

The HiFiGAN framework employs a dual-discriminator setup to ensure the comprehensive authenticity of synthesized audio. The second discriminator in this

setup is the HiFiDiscriminatorS or Scale Discriminator, paired with its ensemble variant, the HiFiDiscriminatorMultiScale.

7.1 The `HiFiDiscriminatorS` Class

- **Operational Mechanism:** The HiFiDiscriminatorS focuses on evaluating the audio signal through a series of 1D convolutions—each layer designed to focus on different frequency bands and temporal resolutions of the signal.
- **Convolution Hierarchy:** It encompasses a cascade of convolutional layers with varying kernel sizes, strides, and groupings, intending to dissect the input signal with increasing granularity.
- **Normalization and Activation:** Each convolutional layer is followed by a leaky ReLU activation, introducing non-linearity into the network's decision-making while being normalized through spectral or weight norms to stabilize the training process.

7.2 The `HiFiDiscriminatorMultiScale` Class

- **Conceptual Foundation:** The Multi-Scale Discriminator encapsulates a set of HiFiDiscriminatorS instances, each operating at a different scale or resolution to capture a wide array of signal distortions and intricacies.
- **Scale-Specific Arbitration:** This ensemble setup amplifies the model's ability to discern details across multiple scales, potentially catching discrepancies invisible to a single scale discriminator.

7.3 Synthesis Judgment and Training

By engaging these two discriminator types in an intricate adversarial dance with the generator, the HiFiGAN setup aims to train a generator so fine-tuned that it can replicate humanly indistinguishable audio textures—an ultimate testament to machine learning's emulation prowess.

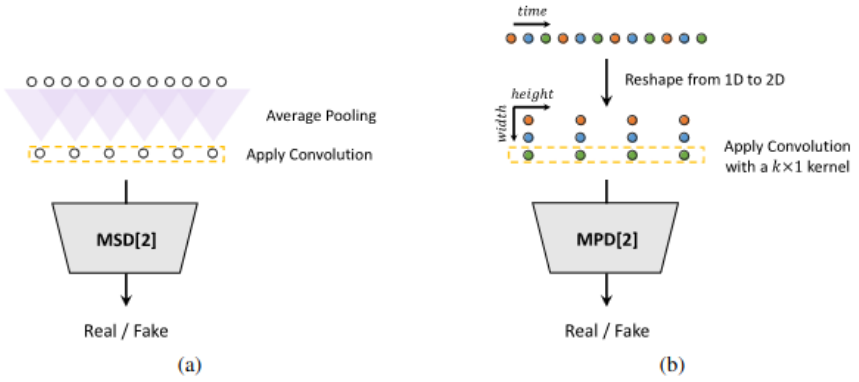


Figure 4: The operational flow within the Scale and Multi-Scale Discriminators

The Scale Discriminators are critical to the High-Fidelity target of HiFiGAN, enforcing a rigorous auditing of the generative output across various auditory facets of detail and quality.

8 Harmonizing Synthesis with HiFiGAN

The HiFiGAN class encapsulates the full generative adversarial network, harmonizing the generator with not one, but two discriminators, in a symphony of neural computations that birth audio outputs of stunning quality.

8.1 The Unified `HiFiGAN` Class

The HiFiGAN serves as the conductor of this sophisticated ensemble, cueing each component at the precise moment to play its part in the audio generation process.

- **The Generator:** At the core sits the HiFiGenerator, an artist wielding neural networks to paint waveforms from spectral canvases of mel spectrograms faithfully.
- **Dual Discriminators:** Complementing the generator are two sets of ears—the HiFiDiscriminatorMultiPeriod and HiFiDiscriminatorMultiScale, each with a keen sense for the nuances of audio. They critique the generator's compositions, pushing for ever-closer approximations of the real deal.
- **Loss Calculation Within Forward Pass:** In a rather innovative twist, the HiFiGAN forward pass doesn't just propagate signals; it also judges them, computing the loss in tandem—and thereby fully encapsulating the learning process.

8.2 Operational Workflow

1. **Audio Generation:** The class begins its act by instructing the HiFiGenerator to transform mel spectrograms into preliminary audio.
2. **Discriminator Evaluation:** The dual discriminators then take the stage, assessing both the real and generated audio to provide feedback. Ingeniously, their evaluation differs depending on whether they focus on generator improvements or discriminator finesse.
3. **Loss Integration:** Finally, the discriminators' feedback culminates in a loss function, which becomes the compass guiding future performances towards more convincing territory.

8.3 Inference Mode

In a separate act, the inference mode, the network foregoes the adversarial critique, focusing solely on rendering the generator's creative prowess into audible content from an input mel spectrogram.

Conclusively, HiFiGAN epitomizes the essence of generative adversarial networks—learning not through isolated toil but through a vibrant interplay of creation and criticism, a paradigm that yields audio not just synthesized, but virtually alive.

9 The Art of Learning: Loss Computation in HiFiGAN

In the training symphony of HiFiGAN, loss functions play first violin, conducting the adjustment of weights with decisive strokes. Let's delve into these functions, the mathematical muse behind the network's learning.

9.1 Feature Matching Loss

- **Harmonizing Features:** The `features_loss` function orchestrates a harmony between the features extracted from the real and synthesized audio by the discriminators, quantifying the difference using the mean absolute error.
- **Intuitive Interpretation:** Each layer of discriminator output contributes to the symphony of features, and the function ensures that the fake outputs closely mirror the composition of the real ones, note by note.

9.2 Generator Loss

The generator's performance is evaluated by `the generator_loss` function:

- **The Critique of Discordance:** This function frames the generator's ultimate goal to create output indistinguishable from reality in the ear of the discriminators.
- **Choreography for Deception:** The discriminators' output serves as a measure of verisimilitude, with the loss encouraging the generator's output to be mistaken by the discriminators for the real deal.
- **Compositional Nuance:** By striving to push the discriminator outputs towards one, the generator is artistically tuning its outputs to the key of authenticity.

9.3 Discriminator Loss

- **Dual Perspectives in Criticism:** `discriminator_loss` evaluates the discriminators' ability to differentiate between the real and fake, rewarding discernment and penalizing confusion.
- **The Duality of Loss:** Seeking an equilibrium, the loss is computed separately for genuine and artificial inputs, then summed up to coax the discriminators towards a balance of skepticism and belief.

9.4 Mel-Spectrogram Loss

Finally, the fidelity of the audio synthesis to the original mel spectrogram is kept in check by `mel_loss`:

- **Anchoring the Tune:** This loss ensures that the generator stays true to the 'sheet music'—the mel spectrogram—by penalizing deviations from the given melodic structure.
- **Keeping the Tempo:** The L1 norm—the difference between each pitch in the predicted and target spectrograms—maintains the generator's tempo, ensuring it composes audio that not only sounds right but follows the right score.

9.5 Conducting the Learning Orchestra

Each loss component plays a critical role in steering the network towards generating waveforms that are not just realistic but also true to the given melodic conditioning. Like a maestro directing a grand musical performance, these losses together shape the learning trajectory of the HiFiGAN towards the highest echelons of audio synthesis.

Final Loss Our final objectives for the generator and discriminator are as

$$\mathcal{L}_G = \mathcal{L}_{Adv}(G;D) + \lambda_{fm}\mathcal{L}_{FM}(G;D) + \lambda_{mel}\mathcal{L}_{Mel}(G)$$
$$\mathcal{L}_D = \mathcal{L}_{Adv}(D;G)$$

Figure 5: A representation of the total loss functions guiding the HiFiGAN's training concert

With the baton of these loss functions, the network diligently refines its parameters, learning to strike the perfect pitch and tempo to echo the original performance.

ectionThe Crescendo of Learning: Total Loss Calculation in HiFiGAN

Calculating the total loss in HiFiGAN represents the crescendo in the model's training, harmonizing all the loss components to fine-tune its generative abilities.

The Composition of Total Loss

- **Adversarial Loss:**
  - The generator's adversarial performance is critiqued twice, once for each discriminator, through the `loss_gen1` and `loss_gen2`. This encourages the generator to craft audio indistinguishable from the real by both scales of judgment.
  - In parallel, `loss_disc1` and `loss_disc2` measure each discriminator's success at accurately detecting real versus fake, refining their discerning ears.
- **Feature Matching Loss:**

  - Feature matching loss is the harmony between the real and fake feature outputs from each discriminator, encapsulated in `loss_fm1` and `loss_fm2`. Doubling this loss accentuates its importance in the model's priority.

- **Mel-Spectrogram Loss:**
  - Ensuring fidelity to the original mel-spectrogram, `loss_mel` penalizes any dissonance between the input and generated melodic structures.
  - This loss is amplified 45 times, emphasizing the model's attention to accurately recreate the melodic contours.

Symphonic Summary

Combining these losses yields `loss_total`, striking a balance between generative creativity and discriminative precision. The ret dictionary not only holds the overall loss but also itemizes each component, documenting the detailed nuances of the network's performance, akin to a setlist of a symphonic performance.

Monitoring Mean Discriminator Outputs

Additionally, the model computes the mean response of each discriminator for both real and fake audio. These insights (`mean_disc1_on_real`, `mean_disc1_on_fake`, etc.) are like audience applause, a real-time feedback for the network's conductor to adjust the ensemble's output in subsequent iterations.

The HiFiGAN Training Performance

With the total loss as its guide, HiFiGAN polishes each note in the generative process. The interplay between the loss components shapes the adversarial training, each iteration bringing the network closer to the virtuoso ability to generate audio of exquisite realism and fidelity.

Final Loss Our final objectives for the generator and discriminator are as

$$\mathcal{L}_G = \mathcal{L}_{Adv}(G;D) + \lambda_{fm}\mathcal{L}_{FM}(G;D) + \lambda_{mel}\mathcal{L}_{Mel}(G)$$
$$\mathcal{L}_D = \mathcal{L}_{Adv}(D;G)$$

Figure 6: Illustration of the total loss computation and its components within the HiFiGAN training cycle.

The model is an opus where the losses are not simply errors but rather the notes that compose the harmony of learning, leading to a glorious crescendo of simulated sound, indistinguishable from the real audio spectrum.

10 Data Collation for Deep Learning: `nv_collate_fn` and `get_collate_fn`

In any machine learning pipeline, preparing the data for the model is as crucial as the model's architecture itself. For batch processing, data collation functions

are the unsung heroes, ensuring that each dataset item fits perfectly into the training batch with the elegance of a puzzle piece finding its right place. We will focus on two such functions tailored for specific datasets in the context of automated speech recognition (ASR) and neural vocoding (NV).

10.1 `nv_collate_fn`

- **The Dynamics of Padding:** The function `nv_collate_fn` assembles a batch from dataset items by padding the tensors to match the longest sequence within the batch. This ensures uniformity, allowing for seamless parallel computations on the batch.
- **Efficient Memory Allocation:** To avoid memory wastage, only the necessary padding is added to each sequence to reach the maximum length in the batch, not a byte more.
- **Length Tracking:** It catalogues the original lengths of the sequences before padding, facilitating operations that may require this information downstream in the pipeline, like masking out the padding in loss computations.

10.2 `get_collate_fn`

- **Dataset-Specific Logic:** This dispatcher function, `get_collate_fn`, serves as a switchboard, linking each dataset with its respective collate function based on the dataset type, ensuring that each one gets the tailored treatment it requires.
- **Adaptive Handling:** For instance, LibrispeechDataset would be paired with an ASR-specific collate function, while LJSpeechDataset or VocoderTestDataset would utilize the NV-specific `nv_collate_fn` outlined above.

10.3 Ensuring Smooth Data Flow

Together, these two functions exemplify the meticulous backstage work of a deep learning system, working diligently to ensure that the spotlighted model can perform without a hitch, singing arias or synthesizing speech with the same ease.

In essence, `nv_collate_fn` and `get_collate_fn` are much akin to the stagehands of a theatrical production, working behind the curtains to set the stage for the performers—the data—to shine in the limelight of the model's computational theatre.

11 The `NVTrainer` Class: Conducting the Neural Vocoder Training

11.1 Initialization and Configuration

The `NVTrainer`, an articulate class inherited from `BaseTrainer`, sets the stage for the training process of neural vocoders. Adapting to the varied cadences of

epoch-based or iteration-based training, this class finely tunes the orchestration of data flow, learning rate adjustments, and the calculated cadence of logging and metric tracking.

11.1.1 Configurable Parameters

Within its constructor, it presents a comprehensive suite of parameters:

- `model`: The neural network model to be trained.
- `metrics`: Criteria for assessing performance.
- `optimizer`: The algorithm that adjusts model weights.
- `config`: A configuration dictionary to manage settings.
- `device`: The hardware destination for tensor computation.
- `dataloaders`: A dictionary housing training and evaluation loaders.
- `lr_scheduler`: An optional argument for learning rate modification.
- `len_epoch`: Defines the length of an epoch based on dataset size.
- `skip_oom`: A flag to bypass out-of-memory errors during training.

11.1.2 Metric Management

The class tracks metrics meticulously:

- A layered `MetricTracker` object holds the progressive learning stages, combining traditional metrics with those bespoke to neural vocoders—like generator and discriminator losses.
- Additionally, it upholds a vigilant watch over gradient norms, ensuring the stability of the learning process.

11.1.3 Batch and Device Management

A static method `move_batch_to_device` diligently ensures that each element of the training batch is transferred to the device at hand, be it GPU or otherwise, for efficient operations.

11.2 Gradual Refinements

`NVTrainer` emphasizes the importance of maintaining the integrity of the gradient's magnitude through `_clip_grad_norm`, an internal method that leverages configuration settings to clip gradients and prevent the infamous exploding gradient problem.

11.3 `_train_epoch` Method

The class equips itself with the `_train_epoch` method as it ventures to encapsulate the intricacies of a training epoch, marking the core steps where the bulk of learning takes place.

11.4 Closing Note

With the `NVTrainer` class, a neural vocoder's training becomes a symphony, each parameter and method a note in a grand composition that produces the harmonious flow of learning and model improvement over time.

11.5 The `_train_epoch` Method: A Training Epoch in Action

The `_train_epoch` method is the heart of the training process, where each epoch unfurls through meticulous iteration over data batches. This method not only drives the learning through forward and backward passes but echoes each step with insightful metric tracking and robust logging, ensuring each epoch's performance is faithfully recorded.

1. **Commence Training**: The model enters training mode, and metrics are reset for a fresh start.
2. **Data Iteration**: Each data batch from the loader is processed with a display of progress thanks to `tqdm`.
3. **Batch Processing**: The batch is sent to `process_batch`, thoroughly prepared and primed for training iteration.
4. **Memory Management**: Encountering out-of-memory errors triggers a graceful skip, releasing gradient memory and clearing cache if so configured by the `skip_oom` flag.
5. **Metric Tracking**: Gradient norms are diligently updated in the tracker after each batch.
6. **Logging Intervals**: At specific intervals, detailed logs are recorded, including scalar values for loss and learning rates, and visual logs for predictions and spectrograms.
7. **Evaluation**: If an evaluation dataloader is provided, the method concludes an epoch with a validation step, often including additional metrics and loss measurements.

Every batch and epoch concludes with a symphonic compilation of metrics—like a musical score, the output is a complex layering of notes (metrics) coming together to describe the performance of the training at that moment.

11.6 Eloquent Epoch Execution

With precision and clarity, the `_train_epoch` method manifests the iterative training process, capturing the essence of learning over iterations with a steadfast rhythm of computations and evaluations, all while being guarded by meticulous metric monitoring and logging.

11.7 Visualization and Audio Logging

Calling into action the chosen metrics and learning artifacts, the method records the synthesized and original audio and mel-spectrograms, offering an auditory account of the model's progress. Each spectrogram and audio snippet is a note in the wider melody of the model's development.

11.8 Concluding with the Transcription of Learning

As the epoch draws to a close, the assembled statistics and insights form a comprehensive log—a testament to the knowledge accrued within the epoch's span, ready to be conveyed back into the flow of training epochs and further refinement.

11.9 Method Details of `NVTrainer`

Taking a closer look at the two pivotal methods of the `NVTrainer` class: `process_batch` and `_evaluation_epoch`. Let's frame each method like a scene in a screenplay, brimming with technical prose.

11.9.1 `process_batch` Method

The `process_batch` method meticulously translates data batches into the language of tensors, moving them onto the device for computation. Each batch is a scene of its own:

1. The scenery is set as the batch make its grand entrance by being ushered onto the computational device.
2. If the curtain is up for training, gradients are wiped clean to make way for the new act.
3. The model, now in the spotlight, performs its forward pass—interpreting audio and mel to produce losses and outputs.
4. Should the outputs form a dictionary, the batch is seamlessly expanded.
5. A switch to the training mode allows the backpropagation, handing the directorial baton to the optimizer to adjust the model's parameters.
6. Should a learning rate scheduler be part of the cast, it adjusts the tempo with a step after the optimizer's move.
7. Finally, the metrics tracker records the ensemble of losses and other performance measures, readying them for the curtain call.

The stage is thus set for learning, with metrics captured for future reference and analysis, preparing the cast for the next scene.

11.9.2 `_evaluation_epoch` Method

Meanwhile, `_evaluation_epoch` audits the model's performance, akin to a dress rehearsal, running through batches without backpropagation:

1. The model, in its evaluation attire, freezes parameters for a consistent performance.
2. As batches parade through the evaluation, they are assessed without the pressure of training, under the watchful eye of metrics that remember each move.
3. Upon completion of the evaluation dance, the metrics are logged and visual insights of the real and synthesized data are complied into tangible artifacts.

Thus concludes an epoch's validation—the meticulous collection of performance snapshots, capturing the essence of the model's progress.

Both the `process_batch` and `_evaluation_epoch` methods form the backbone of the `NVTrainer`'s methodology, exemplifying a seamless blend of computation, error-checking, performance tracking, and optimization that together define the training and validation journey of a neural vocoder.

11.10 The Supporting Acts in `NVTrainer`'s Ensemble

In every training drama, there are crucial supporting characters—the methods and functions—that ensure the lead performance shines. Here's a look at some of the unsung heroes in `NVTrainer`.

11.10.1 The `_progress` Method

The method `_progress` serves up a real-time status snapshot:

1. Computes the current iteration's progress based on batch index and total samples.
2. Elegantly prints out the progress as a percentage of completion.

Its charming simplicity betrays its importance in keeping track of training advances—truly, a glance into the rearview mirror during the journey.

11.10.2 Log Methods: `_log_spectrogram` and `_log_audio`

With `_log_spectrogram` and `_log_audio`, these methods take snapshots of stories told by data:

1. One showcases the spectral landscape—an imaging of sound—via random selection for visual feedback.
2. The other echoes the auditory data back into the world, letting one hear the model's vocal fruits as it learns to sing the data's tune.

Both employ random sampling, allowing for sporadic yet insightful peeks into the batch's compiled chorus.

11.10.3 Monitoring the Model's Strait: `get_grad_norm`

Guarding against the surges in gradient flow is `get_grad_norm`, a vigilant lighthouse:

1. It computes the aggregate norm of gradients, a gauge of how fiercely the neural winds are blowing through the network's parameters.
2. It ensures the network sails smoothly through the tempest of training, uninhibited by the capricious gusts of vanishing or exploding gradients.

Without it, the model's journey might falter in silent numerical storms, unseen but cataclysmic.

11.10.4 The Chronicler: `_log_scalars`

`_log_scalars` - A scribe for performance metrics:

1. Diligently records the averages of tracked metrics, inscribing them in the annals of tensorboard for posterity.
2. Ensures no act of learning, no moment of performance improvement, goes unnoticed or unrecorded.

In the nexus of improvement, this method marks the waypoints, keeping track and time of the learning odyssey.

11.10.5 The Art of Masking: `mask_length`

Lastly, the `mask_length` - a refined touch to the data preparation:

1. Its purpose is concise: zero out the tensor elements beyond the specified lengths.
2. It ensures models focus on the melody of data, not the noise, reinforcing the clean harmony intended in each batch.

Think of it as sheet music with rest notes—where silence is as important as the sound.

11.11 Ensemble in Harmony

Each method plays its part in the training symphony. Whether illuminating the path ahead, capturing the resonant frequencies, gently guiding the model's learning, or recording its tales of growth, the NVTrainer relies on this ensemble of supporting functionality.

12 HiFi-GAN Architectural Overview

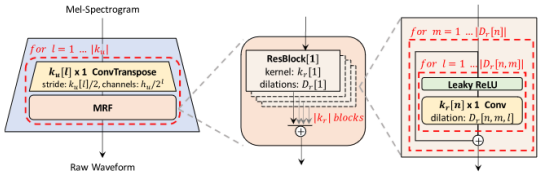


Figure 7: HiFi-GAN Generator Architecture

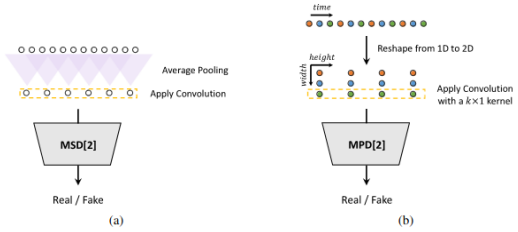


Figure 8: HiFi-GAN Discriminator Architectures

Final Loss Our final objectives for the generator and discriminator are as

$$\mathcal{L}_G = \mathcal{L}_{Adv}(G;D) + \lambda_{fm}\mathcal{L}_{FM}(G;D) + \lambda_{mel}\mathcal{L}_{Mel}(G)$$
$$\mathcal{L}_D = \mathcal{L}_{Adv}(D;G)$$

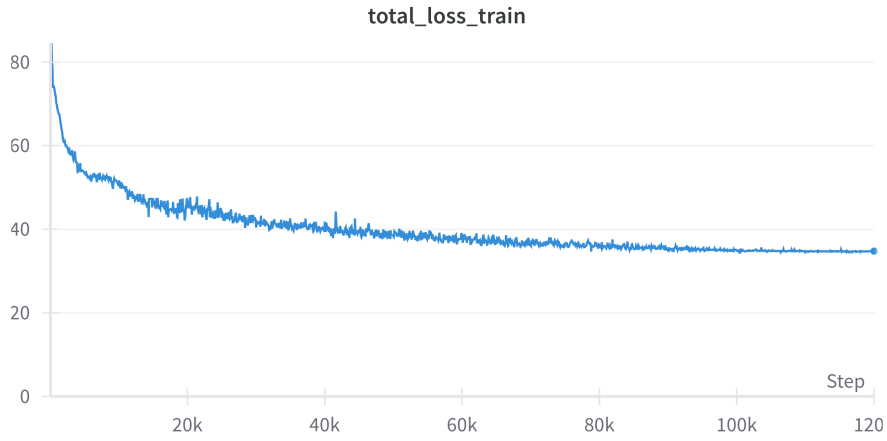
Figure 9: HiFi-GAN Loss Function Components

13 The run before main run

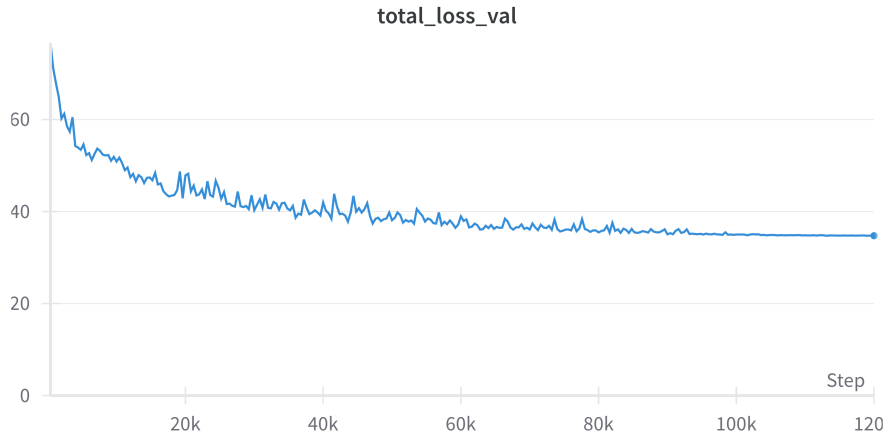
Before the main run, I have trained the Generator on the mel loss alone to ensure that it works. It worked! In short, mel was 0.3

14 Training metrics and loss comparisons of the main run

The duration of that last run was approximately 15 hours. It had 300 epochs, 400 batches per epoch, 16 samples for batch, sequence length 8192 (approximately 370ms with our sample rate 22050Hz)

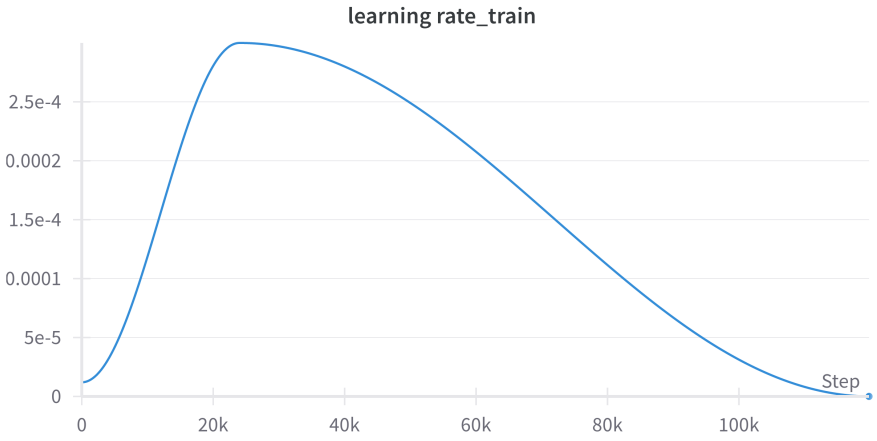


(a) Total Loss during Training



(b) Total Loss during Validation

Figure 10: Total Loss comparison between Training and Validation Phases



(a) Learning Rate over Training
[...]

Figure 11: Various Training and Validation Loss Comparisons

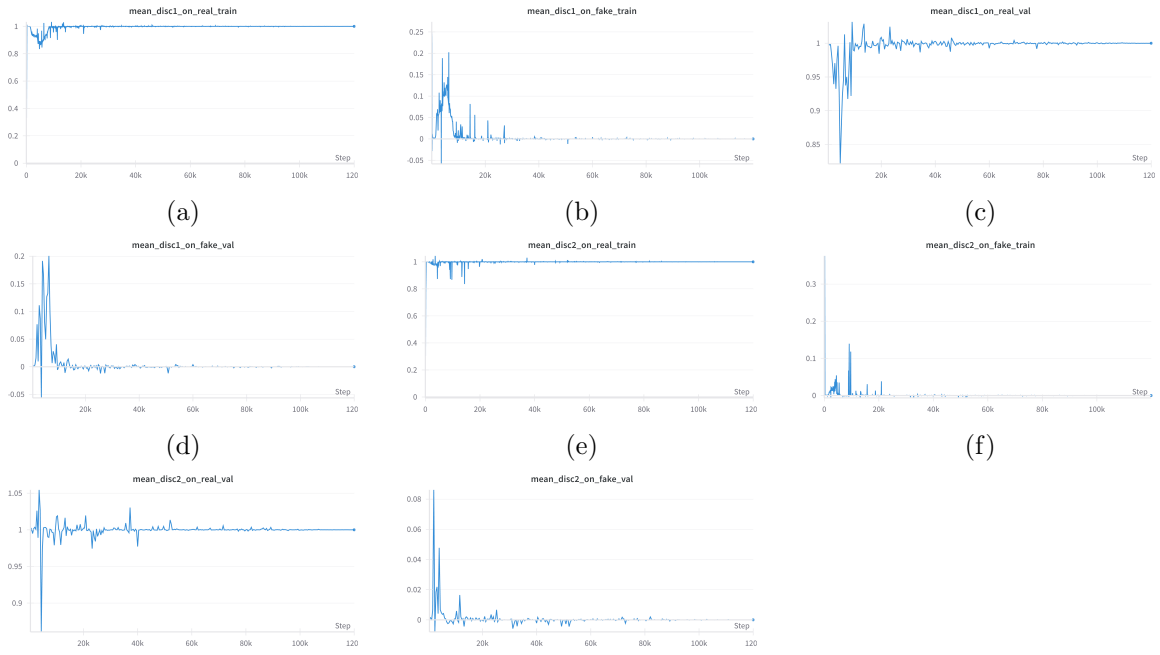
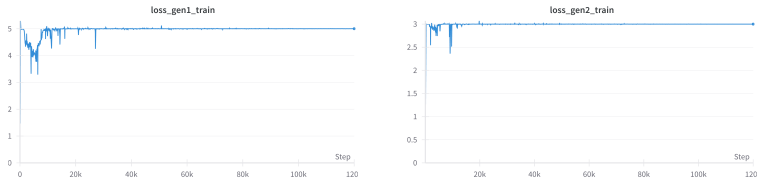
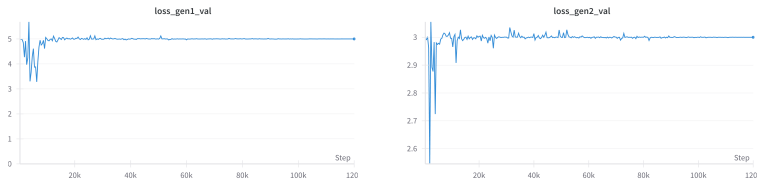


Table 1: Discriminator Mean Outputs on Real and Fake Data during Training and Validation



(a) (b)



(c) (d)

Table 2: Generator Losses on Training and Validation Data

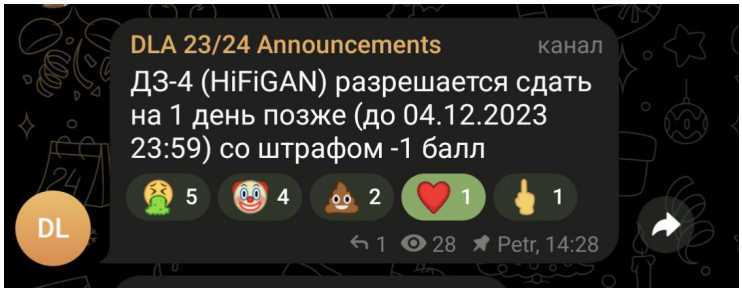
15 WAV synthesis samples, as desired

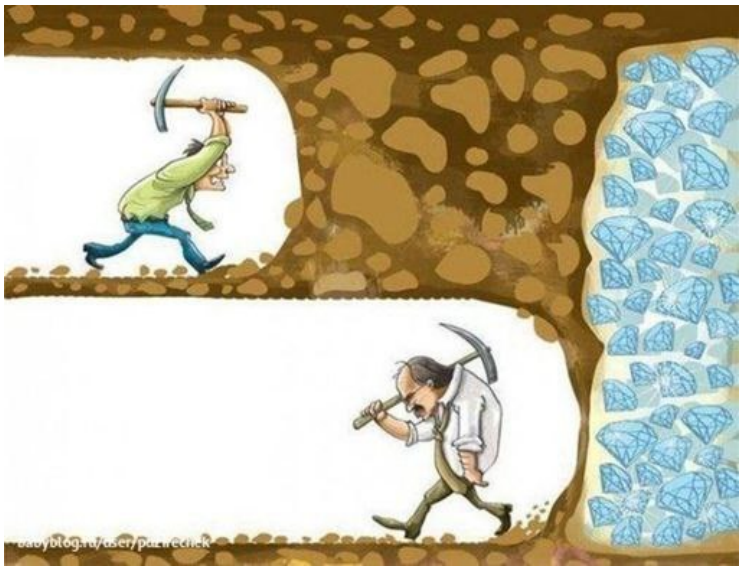
~~Unfortunately, this content is not available in your country~~

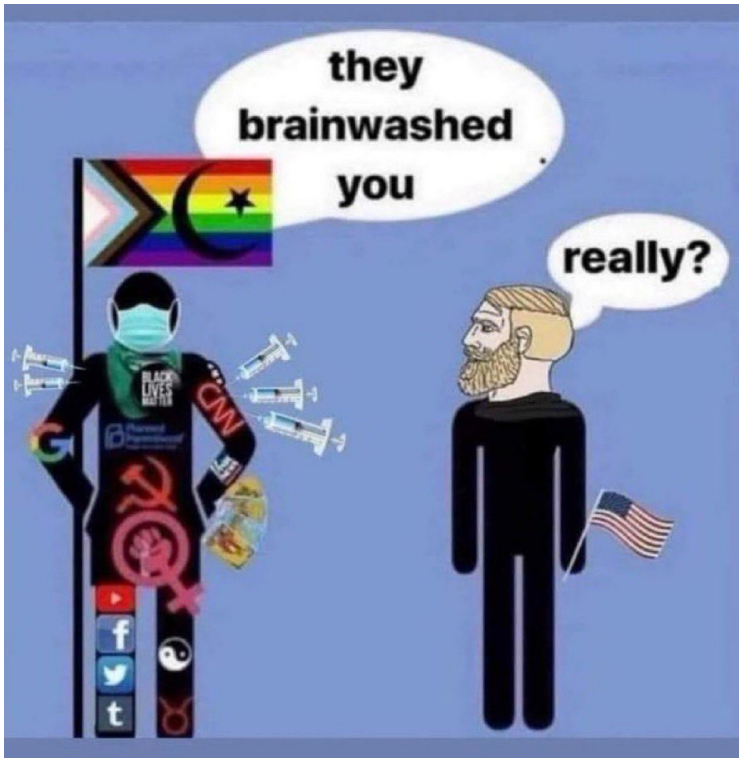
Since the wavs cannot be reliably (cross-platformly) embedded into pdfs, they are available in the repo (check out the readme)

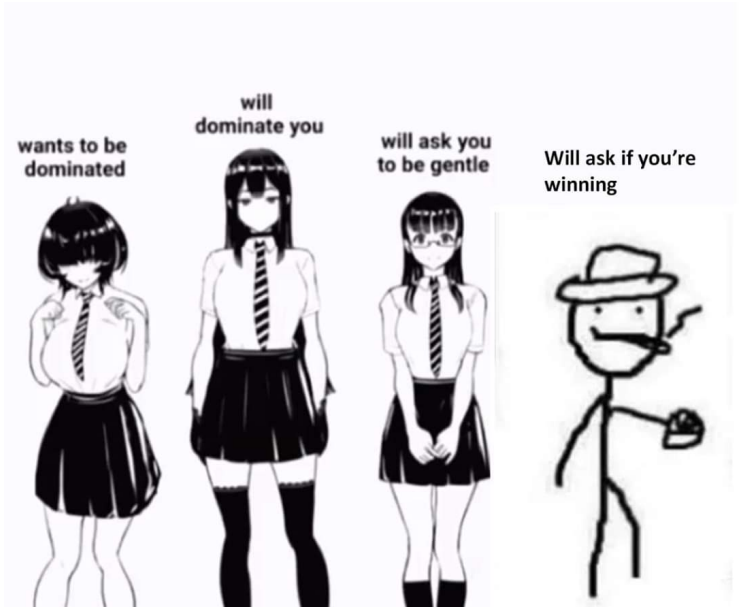
16 Memes and fun content

Since you were able to scroll that far, let me show you some fun content

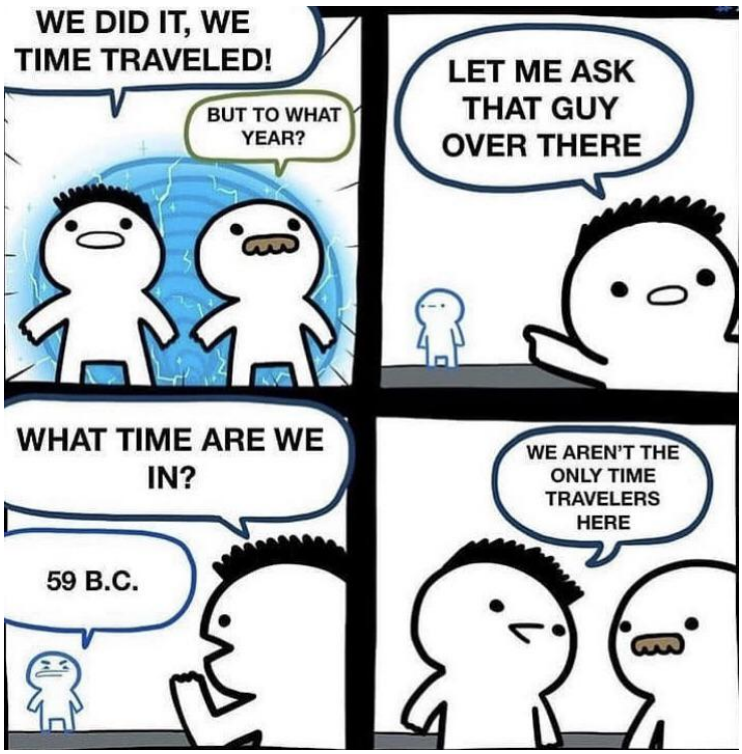




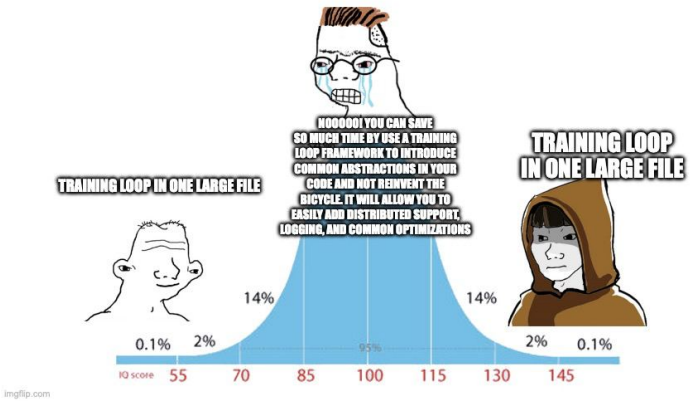

they brainwashed you
really?
BLACK LIVES MATTER
CNN

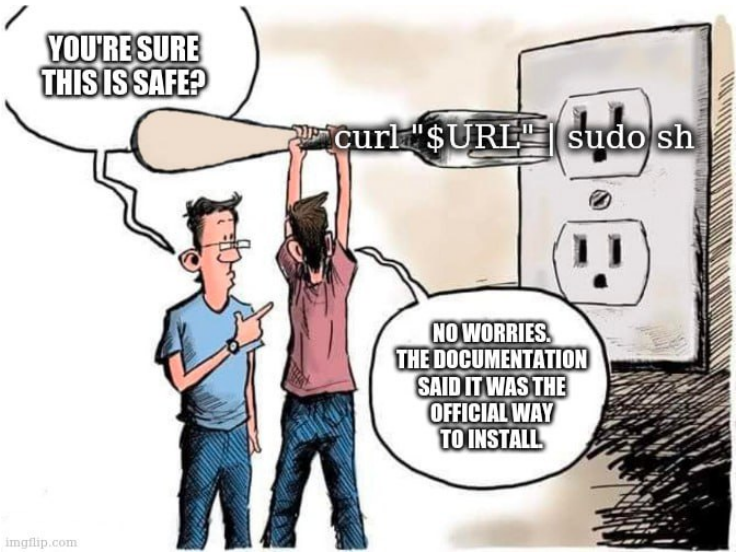

wants to be dominated
will dominate you
will ask you to be gentle
Will ask if you're winning


One of the longest papers on arXiv is extended to over 9000 pages
Academic
arXiv
arxiv.org
Open
24
12
Share
Sort by: Best
Add a Comment
PSquared1234 • 3 yr. ago
This is actually just the author block of any CERN paper.
45
Reply
Share

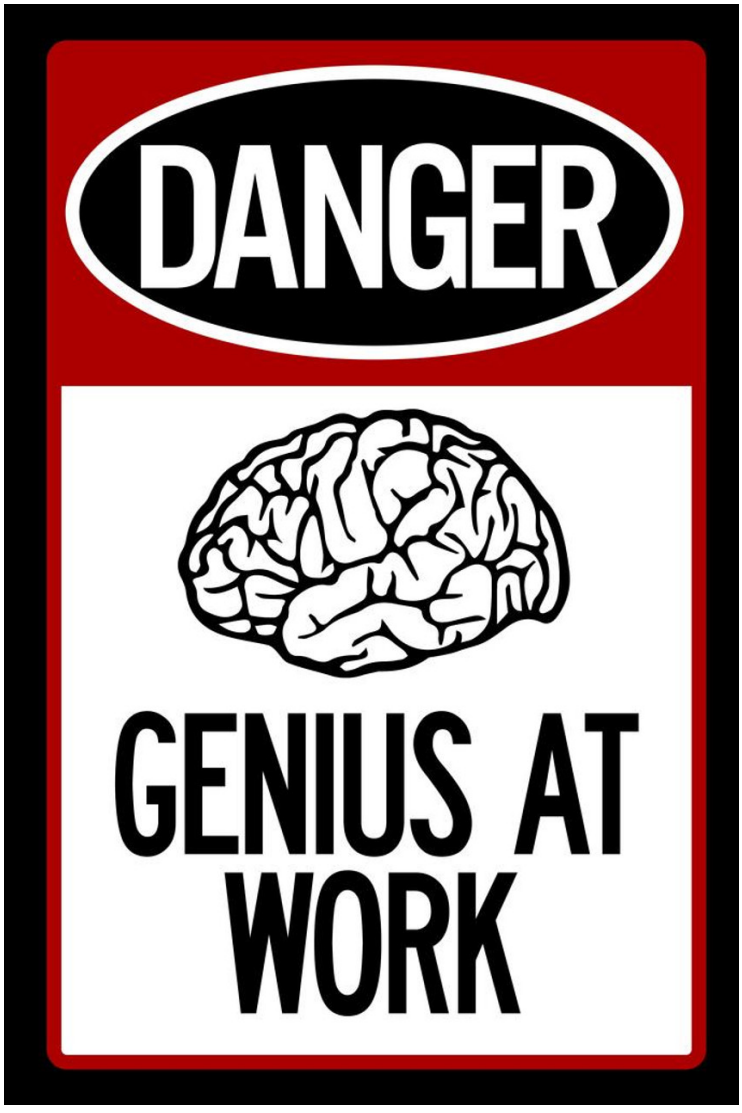

WE DID IT, WE TIME TRAVELED!
BUT TO WHAT YEAR?
LET ME ASK THAT GUY OVER THERE
WHAT TIME ARE WE IN?
59 B.C.
WE AREN'T THE ONLY TIME TRAVELERS HERE

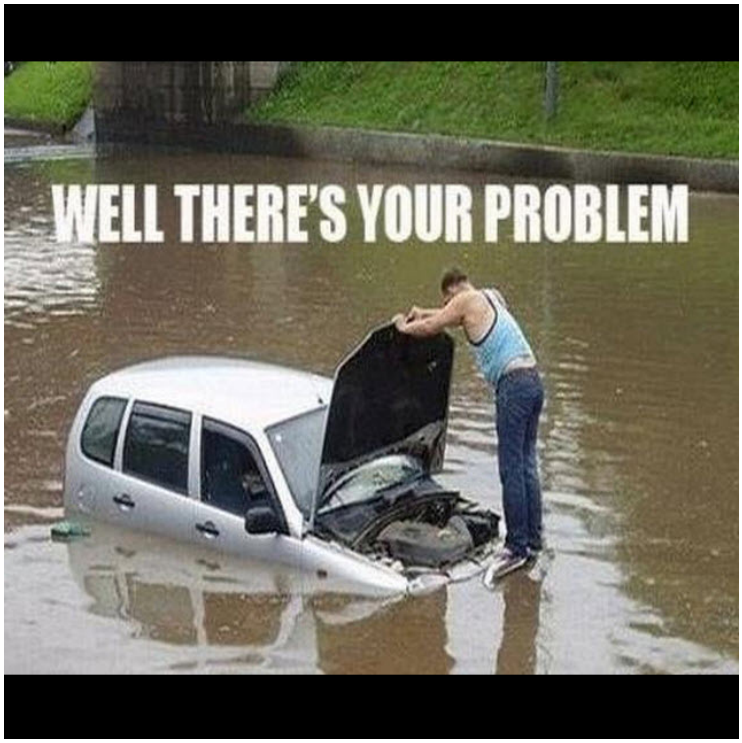

CONFLICTS
CONFLICTS EVERYWHERE
makeameme.org


NOOOOO! YOU CAN SAVE SO MUCH TIME BY USE A TRAINING LOOP FRAMEWORK TO INTRODUCE COMMON ABSTRACTIONS IN YOUR CODE AND NOT REINVENT THE BICYCLE. IT WILL ALLOW YOU TO EASILY ADD DISTRIBUTED SUPPORT, LOGGING, AND COMMON OPTIMIZATIONS
TRAINING LOOP IN ONE LARGE FILE
TRAINING LOOP IN ONE LARGE FILE
0.1%
2%
14%
95%
14%
2%
0.1%
IQ score
55
70
85
100
115
130
145
imgflip.com


YOU'RE SURE THIS IS SAFE?
curl "$URL" | sudo sh
NO WORRIES. THE DOCUMENTATION SAID IT WAS THE OFFICIAL WAY TO INSTALL.
imgflip.com

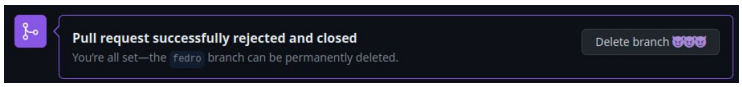

Are ya winning son?
Yes
Oh sorry, looks like I walked into the wrong meme!
It's okay, it happens, but I believe the meme you're looking for is two templates down, right by the "always has been template". You can't miss it.
Alright, got it. Thanks for the help!
No problem king, have a good one.
ifunny.co


DANGER
GENIUS AT WORK


WELL THERE'S YOUR PROBLEM


crypto is a scam, keep money in banks
banks are scams, keep money in crypto


IT IS TIME TO GO.
WAIT
IS MY SON WINNING?
WITH A DAD LIKE YOU?
ALWAYS HAS BEEN


Pull request successfully rejected and closed
You're all set—the fedro branch can be permanently deleted.
Delete branch


SPAM
SPAM


Are ya winning, son ?
Yes, dad. I am.