DeepPlan

> NB: LaTeX here is optimized for Github's Markdown, so please view it on Github. Also, Safari does not render Github's LaTeX and some SVG files well, so Chrome is advised.

Virtually all autoregressive transformer models are trained with the singular objective of next token prediction. They don't possess an explicit objective to think – or better, plan – beyond the next token (though they implicitly do). Here, I present a new transformer model, DeepPlan, that includes an explicit objective of planning beyond the next token, in addition to next token prediction. DeepPlan outperforms, with fewer parameters, a canonical decoder-only transformer, both in train and validation loss.

Motivations

Despite being trained on next token prediction, autoregressive transformer models do develop abilities to plan beyond the next token via the attention mechanism. However, this ability is rather weak, and many failure modes can be attributed to this weakness. Note that I narrowly define planning as anything that happens within a forward pass. Indeed, models can exhibit better planning at the prompt level once you introduce chaining or other clever orchestration logic.

This project explores how planning many steps beyond the next token can be formulated as an objective function during training, in addition to the regular next token prediction. The (perhaps anthropomorphic) intuition is that deliberate planning can improve next token prediction. After all, planning for n future tokens includes the next token. And the objective function formulation is usually the simplest and best way to induce any model behavior.

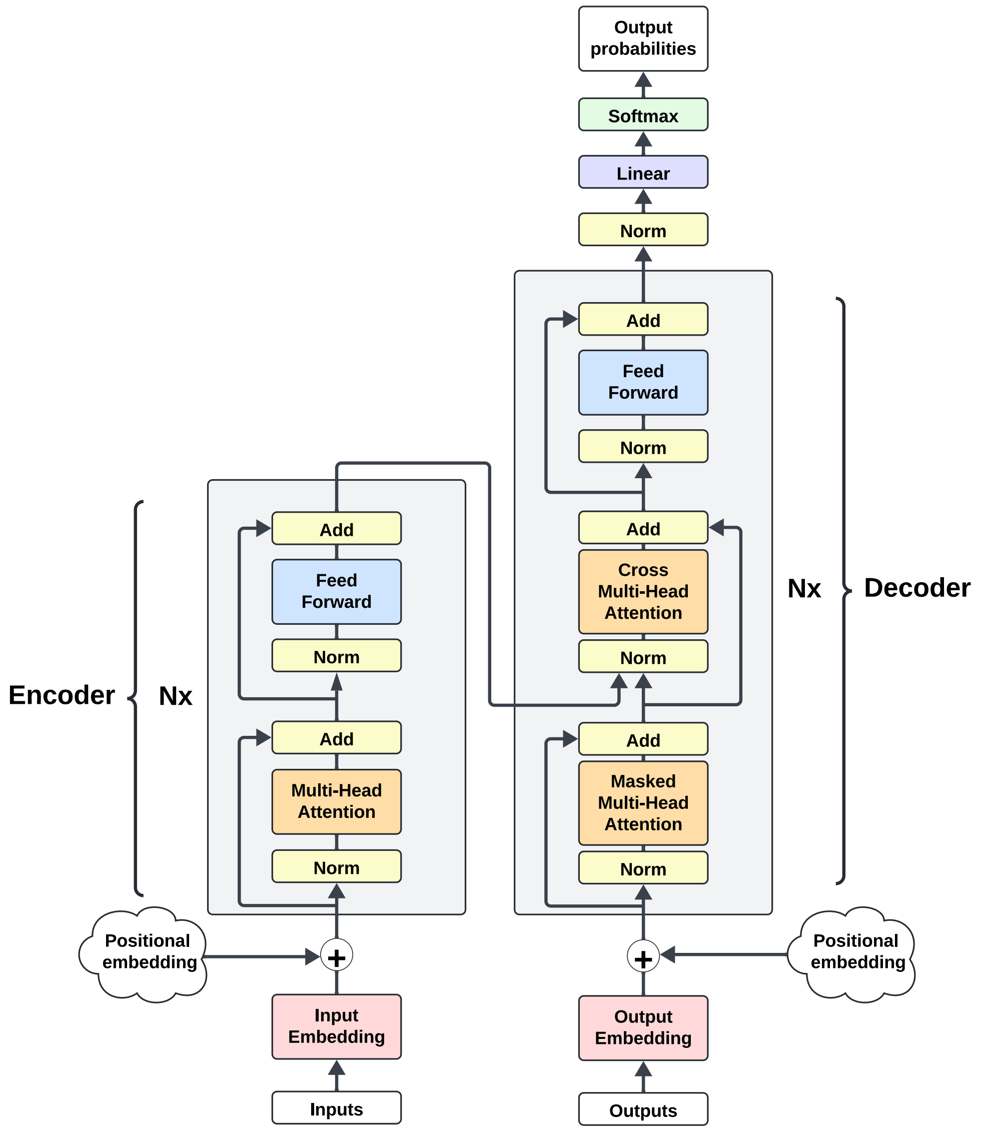
Architecture

At the high level, the architecture consists of an encoder-decoder transformer adapted for end-to-end autoregressive tasks (for those who have read my other model *Auto-regressive Encoder-Decoder Transformer*, it shares the same core architecture but without the positional embedding subtraction). The encoder-decoder separation is necessary for the formulation of the planning objective.

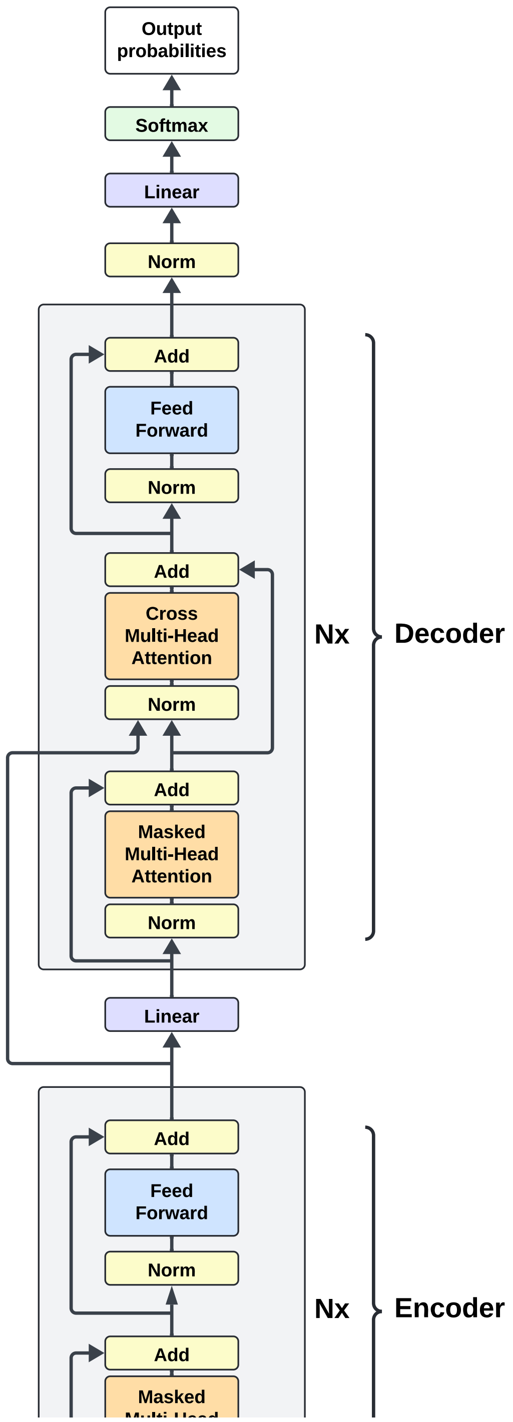
Encoder-Decoder

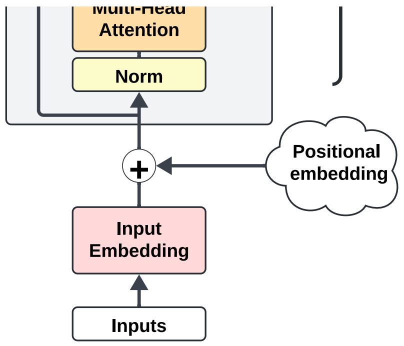
> This section reiterates verbatim the respective section in *Auto-regressive Encoder-Decoder Transformer*. You can skip it if you have already read that one

In the canonical encoder-decoder transformer, the encoder runs once on an input, and the decoder runs auto-regressively on its own output while attending to the encoder output. It looks like the figure below.



To use this architecture for an end-to-end auto-regressive task, the encoder and decoder are adapted to run serially on each new model input. The encoder generates an output and the decoder generates the next token while attending to the encoder output. When a new input is formed with the last decoder output, it gets fed back to the model, which reruns the encoder and decoder. To make this work, the encoder's attention has to be causally masked. The new architecture is shown in the figure below.


Output probabilities
Softmax
Linear
Norm
Add
Feed Forward
Norm
Add
Cross Multi-Head Attention
Norm
Add
Masked Multi-Head Attention
Norm
Nx
Decoder
Linear
Add
Feed Forward
Norm
Add
Masked
Nx
Encoder



Stated alternatively, the new architecture takes a regular decoder-only architecture with L layers and makes the last $L_{decoder}$ layers perform both self-attention and cross-attention on the output of the first $L_{encoder}$ layers.

When transitioning from encoder to decoder, the input to the first decoder layer is generated by a linear pass on the encoder output. For simplicity, the new architecture consists of an equal number of encoder and decoder layers.

Planning loss

To improve the model's planning abilities, an explicit planning objective function must be added. To this end, planning must be first expressed as something that the model can generate. Remember that transformers are excellent at contextual understanding. Normally, the contextual understanding of any hidden state h_t spans the tokens $\{x_i \mid 1 \leq i \leq t\}$. Under this paradigm, the simplest and most natural way to introduce planning is to express it as an extension of understanding that includes future tokens as well. Thus, planning is defined as predicting the latent representation h_t^* that captures the contextual understanding of $\{x_i \mid 1 \leq i \leq t+\delta\}$, where δ is a scalar hyperparameter. Let's denote the context encompassing $\{x_i \mid 1 \leq i \leq t+\delta\}$ the **planning context**, of which $\{x_i \mid 1 \leq i \leq t\}$ is the **present context** and $\{x_i \mid t+1 \leq i \leq t+\delta\}$ is the **future context**. δ is the **future context size** (FCS). Crucially, this planning definition doesn't imply that the future becomes somewhat exposed to the present (e.g. by removing the causal mask in attention), but rather that it becomes part of an objective that the model maximizes.

Next, let's proceed to the three components of any objective function: model output, ground truth, and a minimization function.

On model output, first note that it is hard to make one output – or two closely derived outputs – fulfill two objective functions because different objective functions can cannibalize each other. Hence, a much different output than the one used for next token prediction is preferred. Furthermore, the model must use this output to eventually produce the next token prediction output, so the selected output must play an important role in the next token prediction's computational graph. In a decoder-only transformer, the choice essentially narrows to a hidden state, which is too transient. Alternatively, something more complicated is possible but runs the risk of bloating the model and hindering the gradient flow. However, in an encoder-decoder transformer, there are two naturally distinct outputs, and the decoder attends to the encoder output in every single decoder layer. Moreover, since the encoder focuses more on understanding and the decoder more on predicting the next token, the encoder becomes the natural place for planning to happen. Therefore, the encoder output is selected as the model output of the planning objective function.

Next, generating the ground truth of planning contexts is necessary. Since the encoder output is selected, observe that all the transformations that occur in encoder layers amount to an aggregation of the model input embeddings in a different latent space. This aggregation forms the basis of (present) contextual understanding. Hence, one can expect an affinity between encoder output and a more direct aggregation of the model input embeddings. Given the planning objective function, this affinity can be extended to include future model input embeddings as well. This affinity is precisely what the planning objective function maximizes, or in minimization terms, it minimizes the disaffinity.

Consequently, the ground truth can be generated as planning context embeddings. First, generate present and future context embeddings through cumulative aggregation of model input embeddings. Then, aggregate these two to form planning context embeddings. There are many ways of doing these cumulative aggregations. Here, two different aggregation weightings are used for the present and future. The present context embeddings are cumulatively aggregated with the mean operator. The future context embeddings are cumulatively aggregated with a decaying factor, to reflect the intuition that near-future tokens are easier to predict than distant-future ones. Next, the planning context embeddings average present and future context embeddings. Finally, both encoder output and planning context embeddings are normalized with separate LayerNorm operations, and their disaffinity score becomes the planning loss. Stated more formally,

$\delta :=$ hyperparameter for how many future tokens the model should plan for, inclusive of next token
$out_{enc} :=$ encoder output
$E :=$ model input embedding (detached), comprised of token and positional embedding
$E_{present} :=$ cumulative average of E along T dimension, where $E_{present_{(i,j)}} = \frac{1}{i}\sum_{k=1}^{i} E_{k,j}$

$E_{future} :=$ cumulative aggregation of E along T dimension, where $E_{future_{(i,j)}} = \sum_{k=1}^{\delta} k^{-1} \cdot E_{i+k,j}$

$$E_{plan} = \frac{E_{present} + E_{future}}{2}$$
$$E_{plan\_ln} = LayerNorm_a(E_{plan})$$
$$out_{enc\_ln} = LayerNorm_b(out_{enc})$$
$$planning\_loss = disaffinity\_score(out_{enc\_ln}, E_{plan\_ln})$$

Note that E is first detached because it is used to construct the ground truth. However, because the embedding weights of E are not frozen and thus change, E_{plan} must be re-computed at every forward pass. Perhaps calling E_{plan} "ground truth" is a bit of a misnomer.

Two disaffinity scores are considered. One is mean squared error, and the other is cosine dissimilarity. Cosine dissimilarity is cosine similarity normalized such that zero represents the most similarity and 1 most dissimilarity. So the planning loss with MSE is given by

$$planning\_loss = MSE(out_{enc\_ln}, E_{plan\_ln})$$

and the planning loss with cosine dissimilarity is given by

$$planning\_loss = 1 - \frac{cosine\_similarity(out_{enc\_ln}, E_{plan\_ln}) + 1}{2}$$

A note on E_{future}

Observe the upper bound term δ of

$$E_{future_{(i,j)}} = \sum_{k=1}^{\delta} k^{-1} \cdot E_{i+k,j}$$

When $i > context\_size - \delta$, an index out-of-bounds error will occur. To address this, there are two options. The first is to modify the training code such that the input data X becomes of length $|X| = context\_size + \delta$ tokens but the expected output length remains $|Y| = context\_size$, for each batch. Thus, the additional δ tokens merely serve to satisfy E_{future} but no next token output is expected of them. Consequently, the positional embeddings of the additional δ tokens won't ever be updated because E is detached for the planning loss (this assumes absolute positional embeddings, which are used in this model; it may be different with relative positional embeddings).

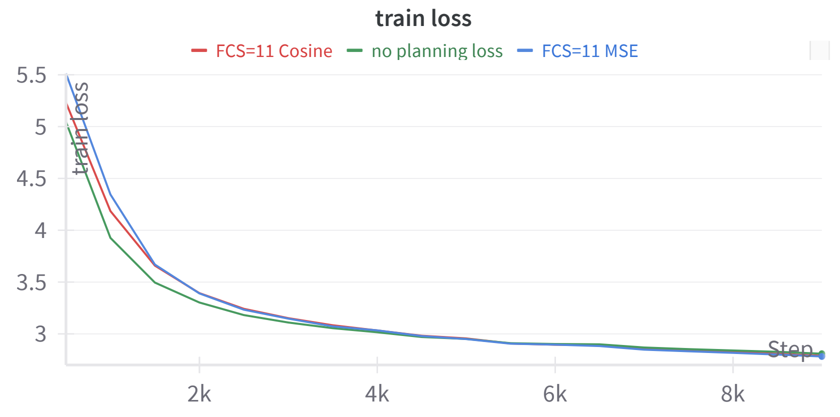
The second option is to just ignore the tokens $\{x_i \mid context\_size - \delta < i \leq context\_size\}$ for the planning loss, meaning there will be only $|context\_size - \delta|$ planning contexts evaluated for the planning loss. Doing so essentially limits the length (i.e. T dimension) of $E_{present}$, E_{future}, E_{plan} and out_{enc} to $|context\_size - \delta|$ in planning loss calculations, thereby restricting the planning optimization scope. This second option is chosen for simplicity.

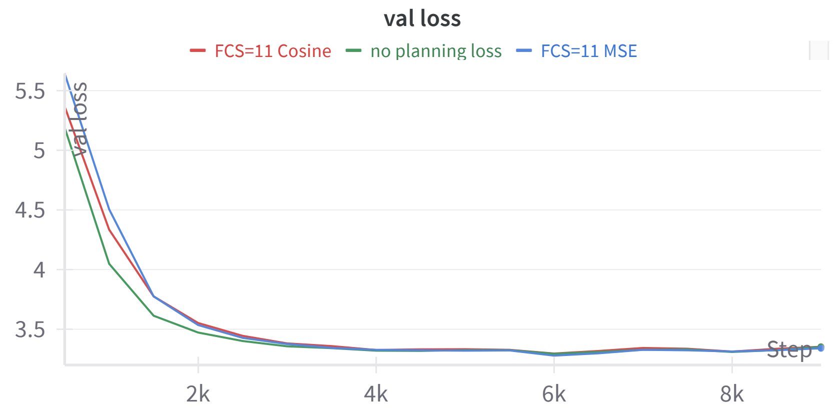
Neither option affects the calculations for the next token prediction loss.

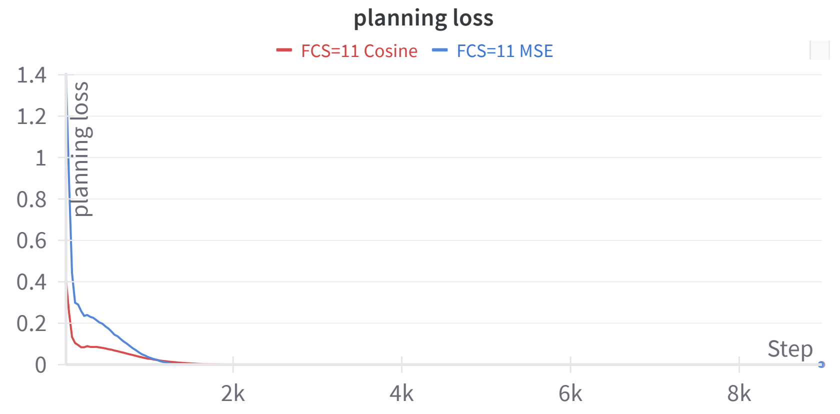
Results

> All training runs below were done on a wikipedia dataset for 9k steps on a single A100 GPU, unless otherwise stated.
>
> Implementation of decoder-only transformer model (baseline) can be found in the `baseline_transformer` directory in this repo

The MSE planning loss outperformed cosine dissimilarity planning loss in both validation and train loss. Both had $\delta = 11$ (chosen arbitrarily). MSE also strictly outperformed an equivalent model without planning loss, and cosine dissimilarity outperformed the same equivalent model in train loss but fell marginally short in validation loss. In general, the planning loss proved beneficial to model performance.







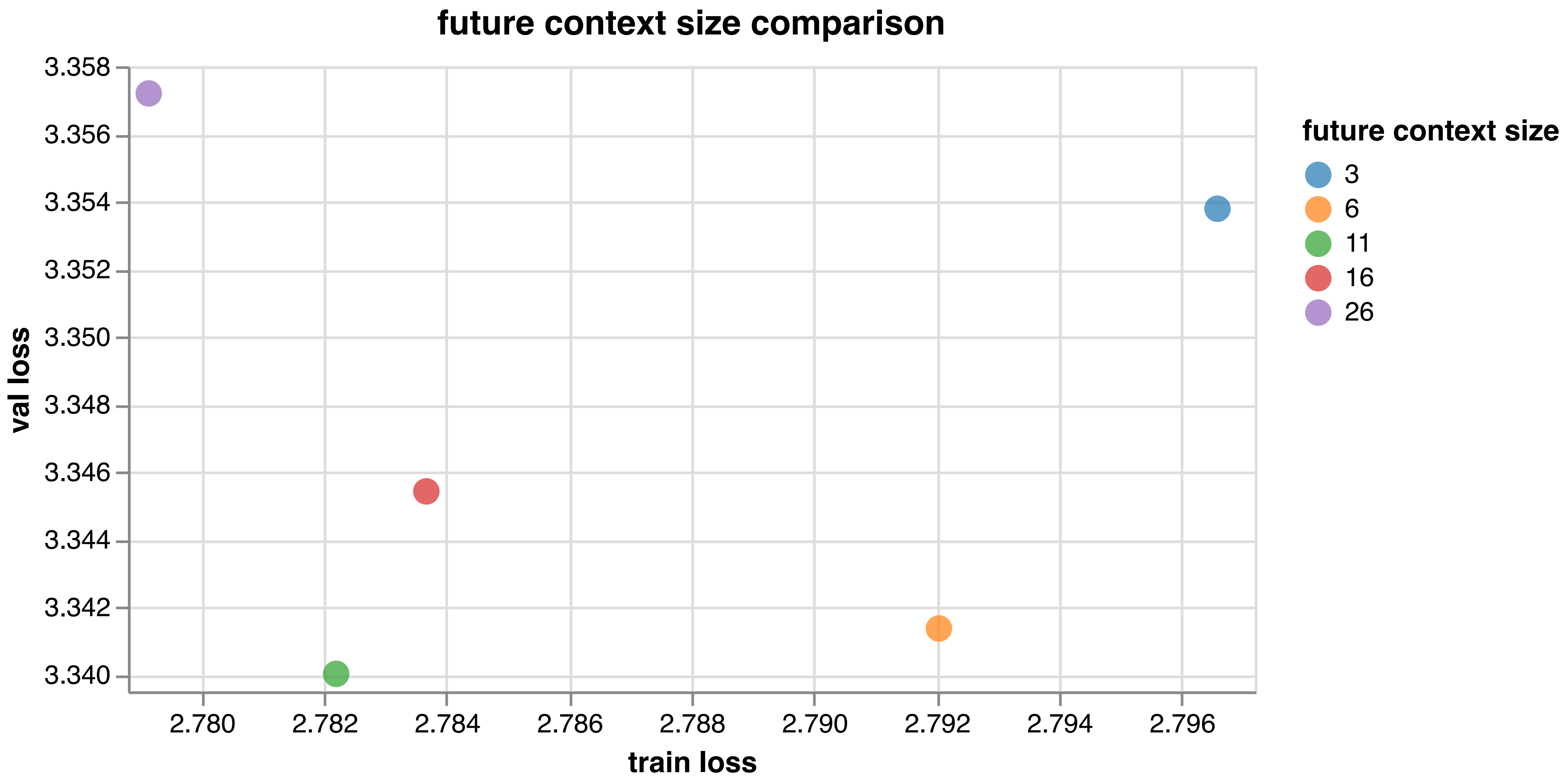
Safari may not render the charts above. Chrome is advised.

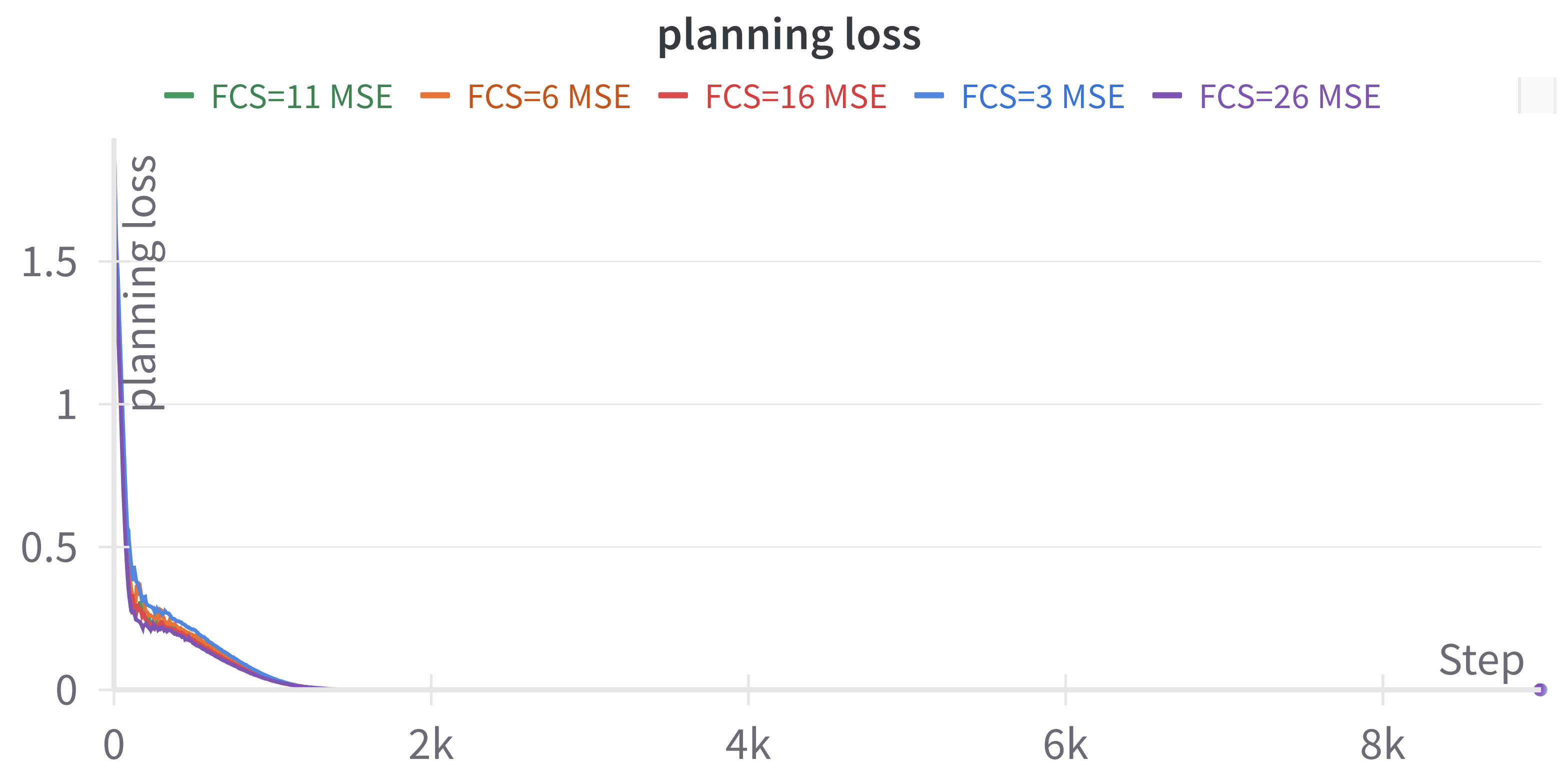
	Train loss	Val loss	Planning loss
FCS=11 Cosine (config)	2.788	3.353	6.999e-9

	Train loss	Val loss	Planning loss
FCS=11 MSE (config)	**2.782**	**3.34**	4.841e-9
no planning loss (config)	2.809	3.352	N/A

Next, using the MSE planning loss, the performances of different δ values were compared. There was a strongly positive correlation between larger δ and better train loss performance. On the other hand, there appeared good positive correlation between larger δ and better validation loss but for smaller δ values only. This narrow correlation can be explained by the second option choice in A note on E_{future}. As δ increases, the planning optimization scope decreases.

"FCS=11 MSE" occupied a strong position in the Pareto frontier, so it was selected for subsequent comparisons.

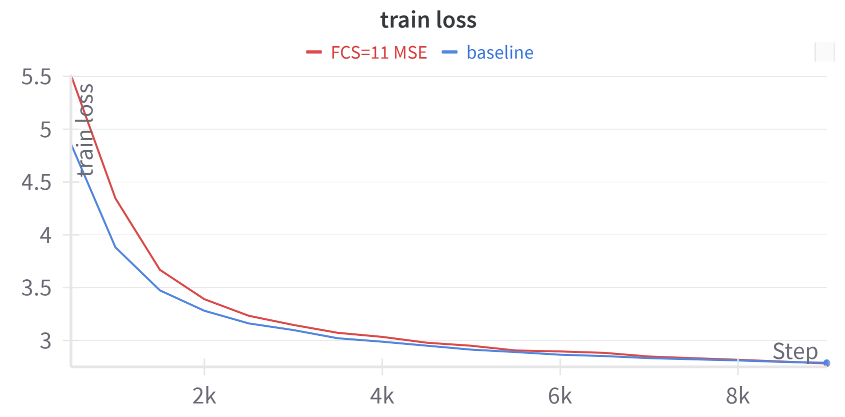

future context size comparison
val loss
train loss
3.358
3.356
3.354
3.352
3.350
3.348
3.346
3.344
3.342
3.340
2.780
2.782
2.784
2.786
2.788
2.790
2.792
2.794
2.796
future context size
3
6
11
16
26

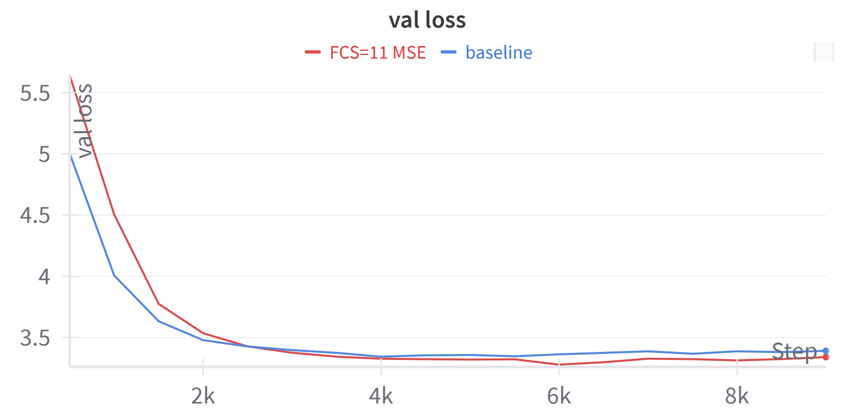


Safari may not render the charts above. Chrome is advised.

	Train loss	Val loss	Planning loss
FCS=3 MSE [(config)]	2.797	3.354	5.855e-9
FCS=6 MSE [(config)]	2.792	3.341	5.082e-9
FCS=11 MSE [(config)]	2.782	**3.34**	4.841e-9
FCS=16 MSE [(config)]	2.784	3.345	5.028e-9
FCS=26 MSE [(config)]	**2.779**	3.357	5.057e-9

Compared to a canonical decoder-only transformer (baseline), the new model outperformed the baseline in both validation and train loss. Both completed in a similar amount of time with similar memory demands, but the baseline had more parameters.

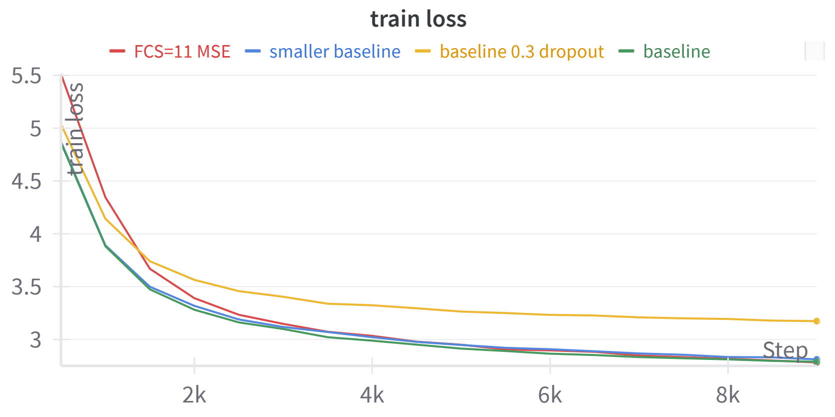


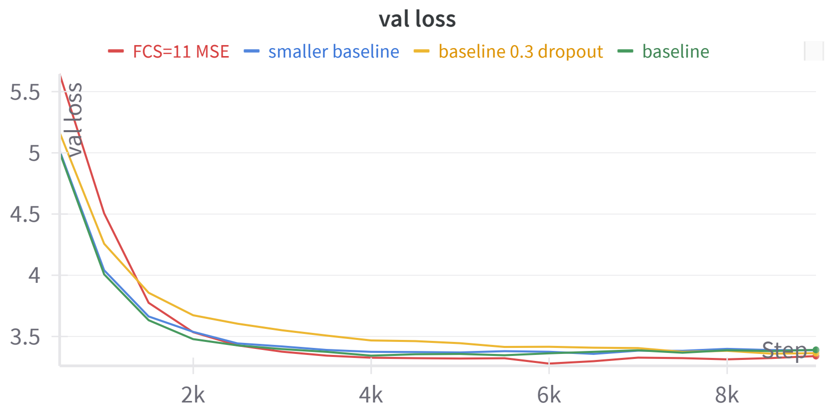


Safari may not render the charts above. Chrome is advised.

	Train loss	Val loss	Size (params)
FCS=11 MSE [(config)]	**2.782**	**3.34**	15,763,500
baseline [(config)]	2.789	3.391	16,036,800

Two more baselines were compared: "smaller baseline" and "0.3 dropout baseline". "smaller baseline" was a baseline smaller (i.e. fewer parameters) than "FCS=11 MSE". Usually, smaller models perform better in validation loss because they overfit less, so the new model's better validation loss could perhaps be explained by its smaller size. By also outperforming "smaller baseline", then the new model's better validation loss can't be attributed to any size explanation. "0.3 dropout baseline" was a baseline with 0.3 dropout. By outperforming it, the new model also demonstrated its superiority over dropout.

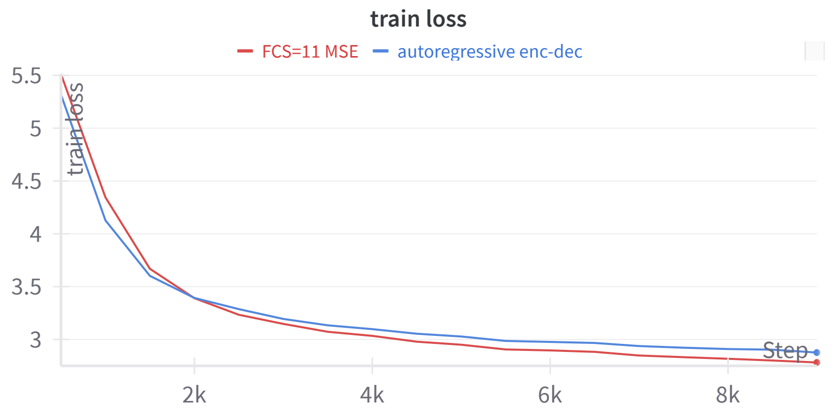


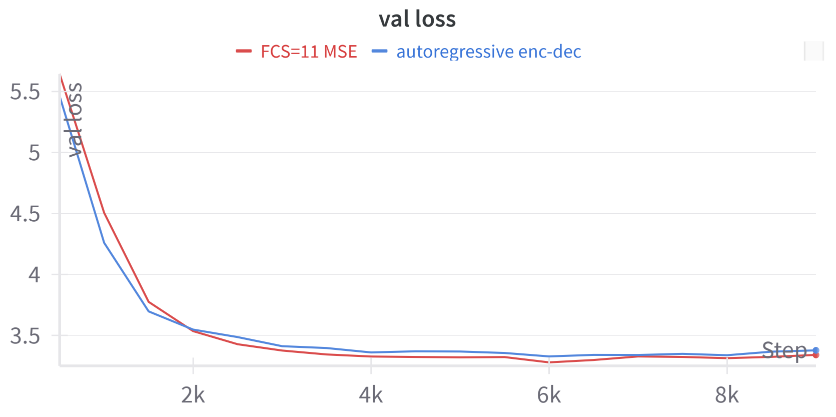


Safari may not render the charts above. Chrome is advised.

	Train loss	Val loss	Size (params)
FCS=11 MSE [(config)]	**2.782**	**3.34**	15,763,500
baseline [(config)]	2.789	3.391	16,036,800
smaller baseline [(config)]	2.811	3.387	15,441,192
0.3 dropout baseline [(config)]	3.173	3.364	16,036,800

Finally, the *Auto-regressive Encoder-Decoder Transformer* model was compared. That model outperformed the baseline in validation loss, and it outperformed the baseline again here. The new model outperformed *Auto-regressive Encoder-Decoder Transformer* in both validation and train loss.





Safari may not render the charts above. Chrome is advised.

	Train loss	Val loss	Size (params)
FCS=11 MSE (config)	**2.782**	**3.34**	15,763,500
autoregressive enc-dec (config)	2.876	3.377	15,763,500

Next steps

These are some further things to look forward to:

- remove the δ hyperparameter and let the model learn the best planning horizon
- explore other ways of constructing present, future, and planning context embeddings, like convolution or even plain matmul. Ideally, the model learns the best aggregation weights
- experiment with unequal encoder and decoder layers, ideally allowing the model to learn the ratio
- instead of MSE and cosine dissimilarity, consider other disaffinity scores
- try bigger models, at least GPT-2 size
- run training for longer to observe long-term behavior

- evaluate on different datasets
- evaluate on non-language tasks
- using relative positional embeddings like Rotary Position Embedding, instead of absolute ones, should make option one in A note on E_{future} more palatable

Conclusions

Given the nature of planning, the hypothesis was that adding a planning objective to a model would improve both train and validation loss. The training loss would improve because of the prominent role that planning plays in the next token prediction. The validation loss would improve because planning shares a strong correlation with generalization. The results from the new model fully substantiate the hypothesis.

Another potential benefit of explicit planning is accelerated inference. A canonical autoregressive transformer only outputs one token at a time, which can be very inefficient. In response, speculative decoding exists to generate multiple future tokens at once. With an explicit latent representation that captures the planning context (and thus the future context), other speculative decoding methods should become possible and perhaps also more efficient & accurate. This is worth exploring.

Alas, the principal limitation is my personal compute budget, so this project cannot avail itself of further analysis and experimentation.

Citation

If you use this codebase, or otherwise found my work valuable, please cite:

```
@misc{yan2024deep-plan,
  title={DeepPlan},
  author={Yan, Yifei},
  year={2024},
  url={https://github.com/yiphei/yif-AI/tree/main/deep_plan_transformer}
}
```

Appendix

Run configs

"FCS=11 Cosine"

```
batch_size: 50
beta1: 0.9
beta2: 0.95
decay_lr: true
est_interval: 500
est_steps: 200
gradient_accumulation_steps: 16
lr: 0.0009
lr_decay_iters: 700000
min_lr: 9.0e-05
model_config:
  context_size: 400
  cross_attn_config:
    n_head: 10
    use_bias: false
  dropout_rate: 0
  future_context_aggregation_type: "DECAY"
  future_context_size: 11
  n_embed: 150
  n_head: 5
  n_layer: 13
  planning_context_ln_type: "POST_AGGR"
```

```
    planning_loss_coeff: 1
    planning_loss_type: "COSINE"
    present_future_context_aggregation_type: "EQUAL"
    use_bias: false
  train_steps: 9000
  warmup_iters: 300
  weight_decay: 0.1
```

"FCS=11 MSE"

```
  batch_size: 50
  beta1: 0.9
  beta2: 0.95
  decay_lr: true
  est_interval: 500
  est_steps: 200
  gradient_accumulation_steps: 16
  lr: 0.0009
  lr_decay_iters: 700000
  min_lr: 9.0e-05
  model_config:
    context_size: 400
    cross_attn_config:
      n_head: 10
      use_bias: false
    dropout_rate: 0
    future_context_aggregation_type: "DECAY"
    future_context_size: 11
    n_embed: 150
    n_head: 5
    n_layer: 13
    planning_context_ln_type: "POST_AGGR"
    planning_loss_coeff: 1
    planning_loss_type: "MSE"
    present_future_context_aggregation_type: "EQUAL"
    use_bias: false
  train_steps: 9000
  warmup_iters: 300
  weight_decay: 0.1
```

"no planning loss"

```
  batch_size: 50
  beta1: 0.9
  beta2: 0.95
  decay_lr: true
  est_interval: 500
  est_steps: 200
  gradient_accumulation_steps: 16
  lr: 0.0009
  lr_decay_iters: 700000
  min_lr: 9.0e-05
  model_config:
    context_size: 400
    cross_attn_config:
      n_head: 10
      use_bias: false
    dropout_rate: 0
    future_context_aggregation_type: null
    future_context_size: null
    n_embed: 150
    n_head: 5
    n_layer: 13
    planning_context_ln_type: null
    planning_loss_coeff: null
    planning_loss_type: "NONE"
    present_future_context_aggregation_type: null
    use_bias: false
  train_steps: 9000
  warmup_iters: 300
  weight_decay: 0.1
```

"FCS=3 MSE"

```
  batch_size: 50
  beta1: 0.9
  beta2: 0.95
  decay_lr: true
  est_interval: 500
  est_steps: 200
  gradient_accumulation_steps: 16
  lr: 0.0009
  lr_decay_iters: 700000
  min_lr: 9.0e-05
  model_config:
    context_size: 400
    cross_attn_config:
      n_head: 10
      use_bias: false
    dropout_rate: 0
    future_context_aggregation_type: "DECAY"
    future_context_size: 3
    n_embed: 150
    n_head: 5
    n_layer: 13
    planning_context_ln_type: "POST_AGGR"
    planning_loss_coeff: 1
    planning_loss_type: "MSE"
    present_future_context_aggregation_type: "EQUAL"
    use_bias: false
  train_steps: 9000
  warmup_iters: 300
  weight_decay: 0.1
```

"FCS=6 MSE"

```
  batch_size: 50
  beta1: 0.9
  beta2: 0.95
  decay_lr: true
  est_interval: 500
  est_steps: 200
  gradient_accumulation_steps: 16
  lr: 0.0009
  lr_decay_iters: 700000
  min_lr: 9.0e-05
  model_config:
    context_size: 400
    cross_attn_config:
      n_head: 10
      use_bias: false
    dropout_rate: 0
    future_context_aggregation_type: "DECAY"
    future_context_size: 6
    n_embed: 150
    n_head: 5
    n_layer: 13
    planning_context_ln_type: "POST_AGGR"
    planning_loss_coeff: 1
    planning_loss_type: "MSE"
    present_future_context_aggregation_type: "EQUAL"
    use_bias: false
  train_steps: 9000
  warmup_iters: 300
  weight_decay: 0.1
```

"FCS=16 MSE"

```
  batch_size: 50
  beta1: 0.9
  beta2: 0.95
  decay_lr: true
  est_interval: 500
  est_steps: 200
  gradient_accumulation_steps: 16
  lr: 0.0009
  lr_decay_iters: 700000
  min_lr: 9.0e-05
  model_config:
    context_size: 400
    cross_attn_config:
      n_head: 10
      use_bias: false
    dropout_rate: 0
    future_context_aggregation_type: "DECAY"
    future_context_size: 16
    n_embed: 150
    n_head: 5
    n_layer: 13
    planning_context_ln_type: "POST_AGGR"
    planning_loss_coeff: 1
    planning_loss_type: "MSE"
    present_future_context_aggregation_type: "EQUAL"
    use_bias: false
  train_steps: 9000
```

```
warmup_iters: 300
weight_decay: 0.1
```

"FCS=26 MSE"

```
batch_size: 50
beta1: 0.9
beta2: 0.95
decay_lr: true
est_interval: 500
est_steps: 200
gradient_accumulation_steps: 16
lr: 0.0009
lr_decay_iters: 700000
min_lr: 9.0e-05
model_config:
  context_size: 400
  cross_attn_config:
    n_head: 10
    use_bias: false
  dropout_rate: 0
  future_context_aggregation_type: "DECAY"
  future_context_size: 26
  n_embed: 150
  n_head: 5
  n_layer: 13
  planning_context_ln_type: "POST_AGGR"
  planning_loss_coeff: 1
  planning_loss_type: "MSE"
  present_future_context_aggregation_type: "EQUAL"
  use_bias: false
train_steps: 9000
warmup_iters: 300
```

```
weight_decay: 0.1
```

"baseline"

```
batch_size: 50
beta1: 0.9
beta2: 0.95
decay_lr: true
est_interval: 500
est_steps: 200
gradient_accumulation_steps: 16
lr: 0.0009
lr_decay_iters: 700000
min_lr: 9.0e-05
model_config:
  context_size: 400
  dropout_rate: 0
  n_embed: 160
  n_head: 10
  n_layer: 26
  use_bias: false
train_steps: 9000
warmup_iters: 300
weight_decay: 0.1
```

"smaller baseline"

```
batch_size: 50
beta1: 0.9
beta2: 0.95
decay_lr: true
```

```
est_interval: 500
est_steps: 200
gradient_accumulation_steps: 16
lr: 0.0009
lr_decay_iters: 700000
min_lr: 9.0e-05
model_config:
  context_size: 400
  dropout_rate: 0
  n_embed: 156
  n_head: 12
  n_layer: 26
  use_bias: false
train_steps: 9000
warmup_iters: 300
weight_decay: 0.1
```

"0.3 dropout baseline"

```
batch_size: 50
beta1: 0.9
beta2: 0.95
decay_lr: true
est_interval: 500
est_steps: 200
gradient_accumulation_steps: 16
lr: 0.0009
lr_decay_iters: 700000
min_lr: 9.0e-05
model_config:
  context_size: 400
  dropout_rate: 0.3
  n_embed: 160
```

```
    n_head: 10
    n_layer: 26
    use_bias: false
  train_steps: 9000
  warmup_iters: 300
  weight_decay: 0.1
```

"autoregressive enc-dec"

```
batch_size: 50
beta1: 0.9
beta2: 0.95
decay_lr: true
est_interval: 500
est_steps: 200
gradient_accumulation_steps: 16
lr: 0.0009
lr_decay_iters: 700000
min_lr: 9.0e-05
model_config:
  add_ln_before_decoder_ff: false
  add_pos_embed_to_decoder: false
  context_size: 400
  cross_attn_config:
    n_head: 10
    use_bias: false
  dropout_rate: 0
  detach_type: 3
  embedding_ln_type: 2
  embedding_loss_coeff: 8
  embedding_loss_type: 2
  n_embed: 150
  n_head: 5
```

```
    n_layer: 13
    order_type: 1
    sub_pos_embed_to_decoder: 2
    use_bias: false
    use_ln_on_encoder_out: true
  train_steps: 9000
  warmup_iters: 300
  weight_decay: 0.1
```