TENSOR PROGRAMS V

Zero-Shot Hyperparameter Transfer

Tuning Large Neural Networks via Maximal Update Parametrization

Greg Yang, Edward J. Hu, et al.

Microsoft & OpenAI

PRESENTATION OVERVIEW

Contents

THE PROBLEM

The Hyperparameter Tuning Challenge

Why HP Tuning is Critical

Poor HPs → Subpar Performance
Bad hyperparameters cause training instability and reduced model quality

Baseline Comparability Issues
Varying HP tuning makes published results hard to compare fairly

Scaling Exacerbates the Problem
Billions of parameters make tuning prohibitively expensive

The µTransfer Solution

Core Idea: Tune HPs on a small proxy model, then zero-shot transfer them to the full-size target model without direct tuning.

The Problem: Optimal Learning Rate Shifts with Width (Standard Parametrizatio n)

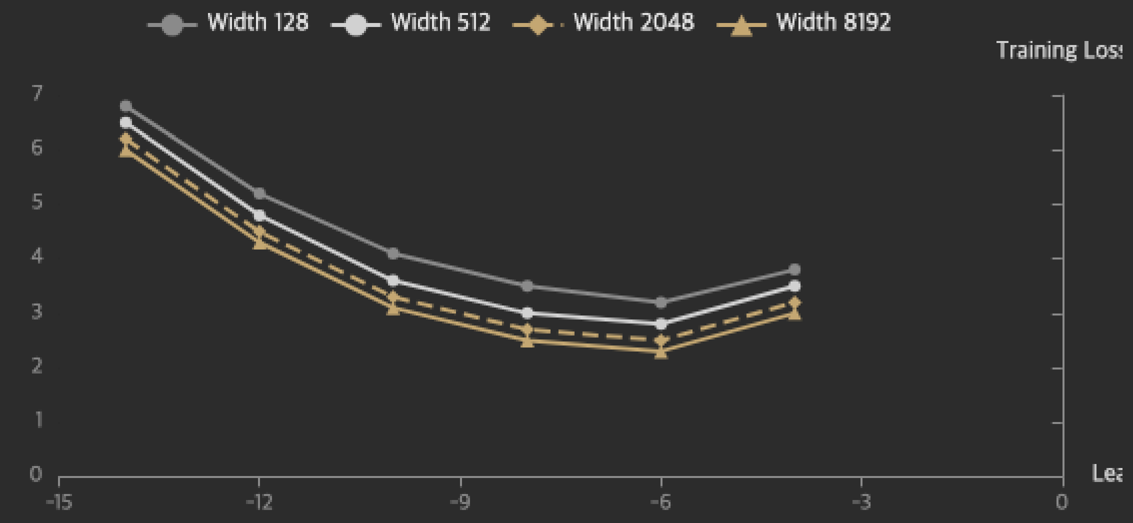


Figure 1 (Left): In standard parametrization, optimal learning rate shifts by orders of magnitude as Transformer width i ncreases. Using small model HPs on large models causes divergence.

13M → 350M
BERT-large
Outperformed published num bers

40M → 6.7B
GPT-3
Only 7% tuning cost

Key Innovation
µP
Enables HP stability

THEORETICAL FOUNDATION

Why Parametrization Matters: A Mathematical Primer

Central Limit Theorem Analogy

The Central Limit Theorem (CLT) provides the foundation for understanding correct parametrization in neural networks.

Given iid samples $x_1, \ldots, x_n \sim N(0, 1)$:

$$1/\sqrt{n}\ (x_1 + \cdots + x_n) \to N(0, 1)$$

The scaling factor **$1/\sqrt{n}$** is the "correct" order for convergence.

$c_n = 1/n$	$c_n = 1$	$c_n = 1/\sqrt{n}$
$\to 0$	$\to \infty$	$\to N(0,1)$
Shrinks to zero	Blows up	Converges

Extension to Multiple HPs

For multiple hyperparameters $c_1, \ldots, c_k$:

$$F_n(c_1,\ldots,c_k) = E[f((c_1+\cdots+c_k)(x_1+\cdots+x_n))]$$

The correct reparametrization is:

$$\alpha_i = c_i\sqrt{n}$$

Critical Insight: All HPs must be correctly parametrized. Fixing some while neglecting others causes compensation distortions.

Connection to Neural Networks

Neural network training parallels this optimization problem. Correct parametrization enables direct HP transfer from narrow to wide networks.

Standard Parametrization (SP)

- Initialization: $\Theta(1/\text{fan\_in})$
- Learning rate: $\Theta(1)$

❌ No well-defined infinite-width limit

Maximal Update Parametrization (μP)

- Input: $\Theta(1/\text{fan\_out})$
- Hidden: $\Theta(1/\text{fan\_in})$
- Output: $\Theta(1/\text{fan\_in}^2)$

✓ Enables feature learning

Key Theoretical Result:

μP is the unique parametrization that allows HP transfer across width while preserving feature learning in the infinite-width limit.

Core Principle

If we parametrize correctly, optimal HPs from a narrow network transfer approximately

EMPIRICAL EVIDENCE

Hyperparameter Instability in Standard Parametrization

MLP on CIFAR-10: Optimal Learning Rate Shifts

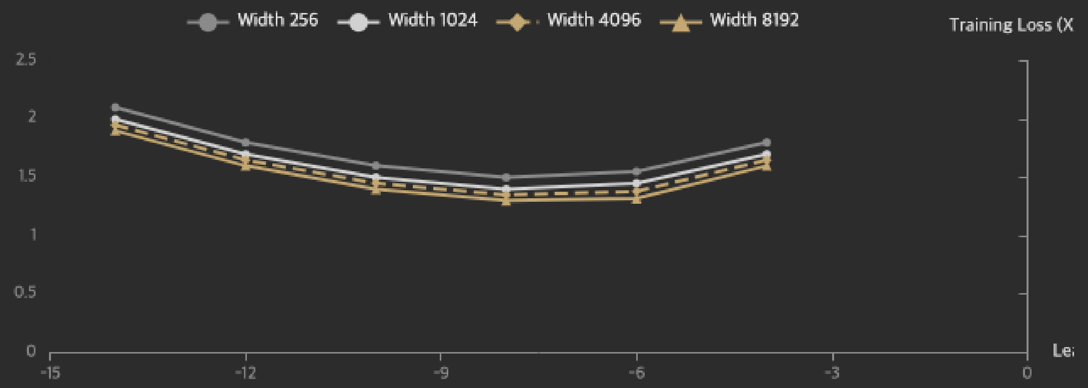
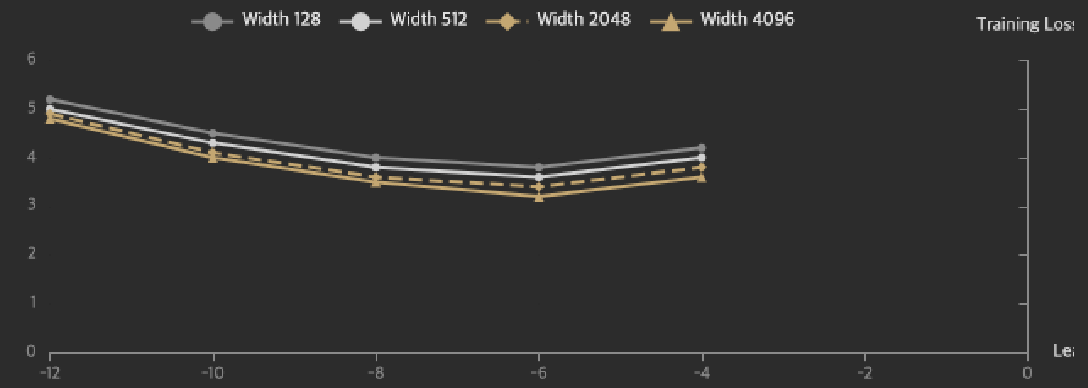


Figure 3: MLP with 2 hidden layers, ReLU activation, SGD optimizer. Optimal LR shifts ~4x (2 orders of magnitude in log space) as width increases from 256 to 8192.

Optimal LR
-12 → -6

Transformer on Wikitext-2: Same Instability



The Consequences

1 **Divergence Risk**

Using small model optimal LR on large model causes divergence or poor performance.

2 **No Transfer Guarantee**

HP instability means each model size requires expensive independent tuning.

3 **Beyond Learning Rate**

Initialization scale and other HPs also shift with width (Fig. 18).

Experimental Setup Details

MLP

- 2 hidden layers
- Width: 256 → 8192
- Activation: ReLU
- Loss: Cross-entropy

Transformer

- d_model: 128 → 8192
- Depth: 2
- Optimizer: Adam
- Dataset: Wikitext-2

THE SOLUTION

Maximal Update Parametrization (μP)

What is μP?

The Maximal Update Parametrization (μP) is a principled scaling of initialization and learning rates that ensures each layer is updated on the same order during training, regardless of width.

Key Design Principle:

All hidden activations update at the same speed in terms of width. No layer updates too fast or too slow as network scales.

Standard Parametrization

- Layers update at different speeds
- Output blows up with width
- No HP transfer possible

Maximal Update Parametrization

- Uniform update speed across layers
- Stable activations with width
- Enables zero-shot HP transfer

μP Solves the Problem: Stable Optimal LR

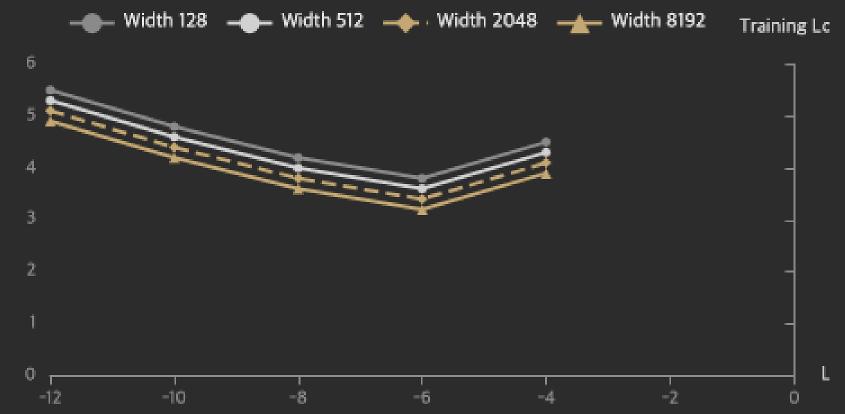


Figure 1 (Right): In μP, optimal learning rate is stable across all widths. Best LR for width-128 is also best for width-8192. Performance improves monotonically with width.

μP Scaling Rules (Table 3)

Input Weights & All Biases
Init Var: 1/fan_out | Adam LR: 1

Output Weights
Init Var: 1/fan_in | Adam LR: 1/fan_in

Hidden Weights
Init Var: 1/fan_in | Adam LR: 1/fan_in

Transformer Addition: Attention logits scaled by $1/d$ instead of $1/\sqrt{d}$

Why "Maximal Update"?

μP enables "maximal" feature learning in the infinite-width limit:

- Each layer contributes meaningfully to updates
- Activations don't vanish or explode
- Preserves feature learning (unlike NTK limit)

Contrast with NTK: Neural Tangent Kernel parametrization loses feature learning ability in infinite width. μP preserves it, making it suitable for modern large-scale pretraining.

MECHANISM

How µP Works: Defects of SP and µP Fixes

The Blow-Up Problem in Standard Parametrization

In SP, network output blows up with width after just **1 step of SGD** due to imbalanced layer updates.

1-Hidden-Layer Linear Perceptron Example:

$$f(x) = V^T U x$$

where $U, V \in \mathbb{R}^{n \times 1}$ (width n)

SP Initialization:

$V \sim N(0, 1/n)$, $U \sim N(0, 1)$

After 1 SGD step (LR=1):

$f'(x) = (V^T U + \theta U^T U + \theta V^T V + \theta^2 U^T V)x$

The Problem:

$U^T U = \Theta(n)$ by Law of Large Numbers

Output blows up linearly with width n

µP Prevents Blow-Up

Same network in µP:

µP Initialization & LR:

$V \sim N(0, 1/n^2)$, $U \sim N(0, 1)$

$\eta\_V = 1/n$, $\eta\_U = n$

Empirical Evidence: Transformer Activations

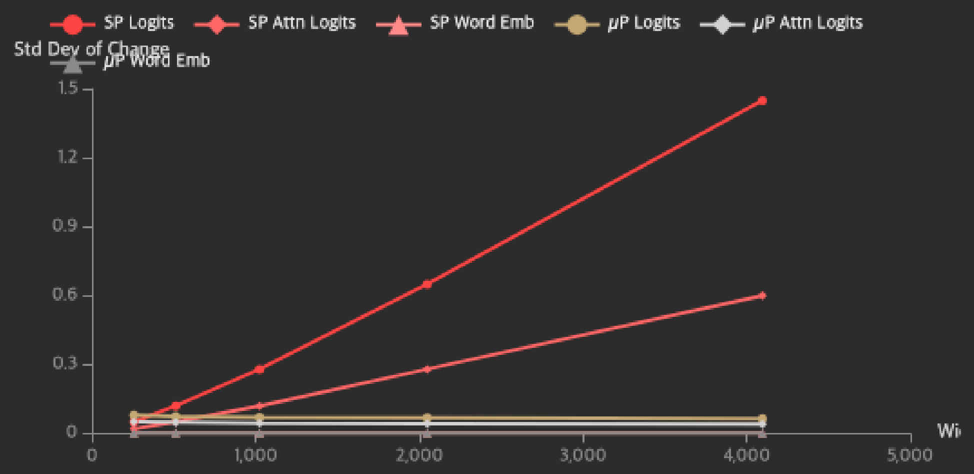


Figure 5: Standard deviation of coordinate changes from initialization over 4 Adam steps. Logits and attention logits blow u p in SP but remain stable in µP.

Why Scaling Down LR in SP Fails

Simply reducing learning rate with width doesn't work:

- ✕ **Word embeddings** update by width-independent amount per step. Small LR means the y don't learn in large models.
- ✕ **Input vs output layers** have different scaling. SP updates them at different rates.

TRANSFERABILITY

Which Hyperparameters Can Be µTransferred?

Three Categories of Hyperparameters

Category 1: Transferable

These HPs can be transferred from small to large models:

- Learning rate
- Momentum
- Adam β_1, β_2
- LR schedules
- Initialization variance
- Parameter multipliers
- Per-layer HPs

Category 2: Not Transferable

Regularization HPs — their purpose depends on both model and data size:

- Dropout probability
- Weight decay
- Data augmentation
- Label smoothing

Category 3: Transferred Across

Scale-defining HPs across which others are transferred:

- Width
- Depth
- Batch size
- Sequence length
- Training steps

Empirical Validation: HP Stability Across Width & Depth

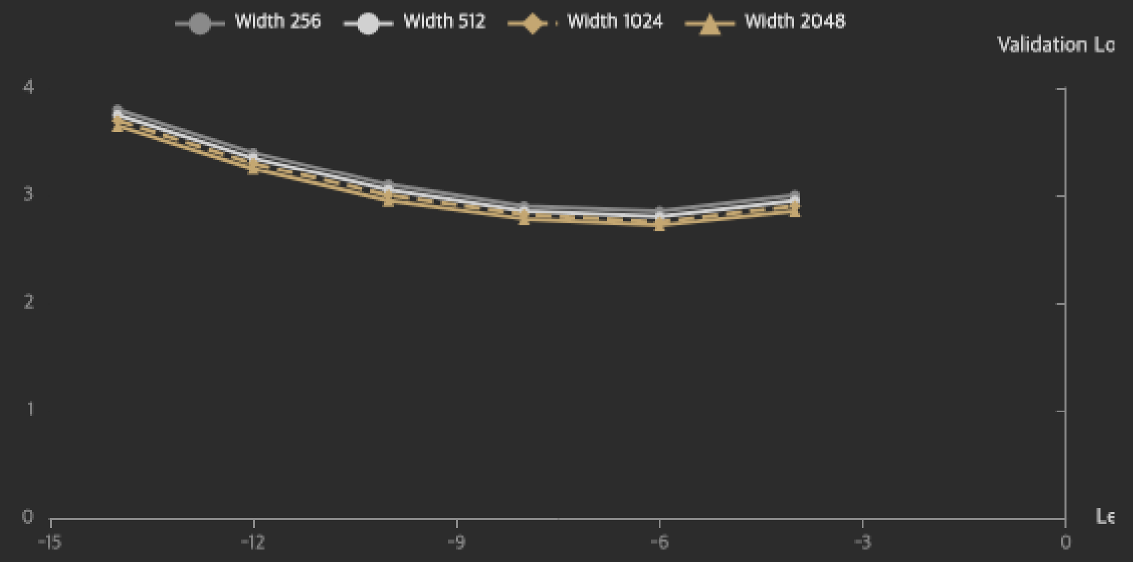


Figure 4: Four representative HPs on pre-LN Transformers in µP: learning rate, output weight multiplier α_output, initialization std σ, and LR schedules (constant, linear, cosine, etc.). Models trained on Wikitext-2 for 10k steps.

Transfer Requirements

For stable transfer, ensure:

- Width ≥ 256
- Depth ≥ 4
- Batch size ≥ 32
- Seq length ≥ 128

Limitations

- Init std doesn't transfer well across depth
- Depth transfer only works for pre-LN
- Optimum can shift slightly at scale

→ But shift has small impact vs SP

ALGORITHM

The µTransfer Algorithm

Algorithm 1: µTransfer

1 Parametrize Target Model

Parametrize the target (large) model in Maximal Update Parametrization (µP)

2 Tune Proxy Model

Tune a smaller version (in width and/or depth) of the target model

3 Copy Hyperparameters

Copy tuned hyperparameters to the target model

Result:

Near-optimal HPs on the full-sized target model **without directly tuning it at all!**

Key Benefits

- **Massive Speedup:** Tuning cost independent of target model size
- **Tune Once:** Single proxy tune serves whole model family
- **Better Performance:** Outperforms SP with optimal LR

Large-Scale Results: BERT & GPT-3

BERT (350M params)

Proxy: 13M model

Tuning cost = 1 BERT-large pretrain

Test Loss:

Megatron: 1.683

µTransfer: 1.731 → Better

GPT-3 (6.7B params)

Proxy: 40M model

Tuning cost: only 7% of pretraining

Validation Loss:

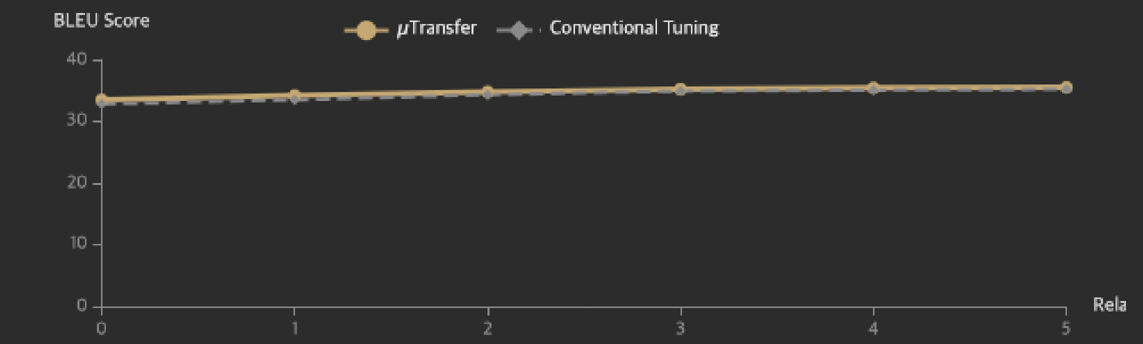
Original [7]: 2.03

µTransfer: 1.98 → Better

Revolutionary Efficiency:

For models like GPT-3, HP tuning is not feasible at all without µTransfer. The approach reduces tuning cost from 100% to 7% of pretraining.

Compute-Performance Pareto Frontier



LARGE-SCALE VALIDATION

Experimental Results: BERT and GPT-3

BERT Pretraining Results

Setup:

- Proxy: 13M params
- Target: BERT-base (110M) & BERT-large (350 M)
- Tuning: 256 HP samples
- Training: 10^5 steps
- Cost ≈ 1 BERT-large pretrain
- No direct tuning of targets

BERT-base Results:

Method	Test Loss	MNLI (m/mm)
Megatron	1.995	84.2/84.2
Naive	diverged	—
μTransfer	1.970	84.3/84.8

BERT-large Results:

Method	Test Loss	MNLI (m/mm)
Megatron	1.683	86.3/86.2
Naive	diverged	—
μTransfer	1.731	87.0/86.5

GPT-3 6.7B Results

Setup:

- Target: 6.7B model
- Proxy: 40M model
- Size ratio: 168×
- Tuning cost: 7% of pretrain
- Used FP32 (vs FP16 baseline)
- Relative attention

Key Evaluation Results:

Validation Loss	1.98 < 2.03
PTB Perplexity	11.4 < 13.0
LAMBADA Zero-Shot	73.5% > 70.8%
HellaSwag Zero-Shot	72.0% > 66.7%

Remarkable Achievement:

μTransferred 6.7B model performs comparably to the twice-as-large 13B model from [7], despite using FP32 and encountering numerical issues.

Practical Impact

- **Cost Reduction:** From prohibitive to 7% of pretraining cost
- **Time Savings:** 220× speedup for BERT-large tuning
- **Performance:** Matched or exceeded published baselines

INSIGHTS

Key Properties and Theoretical Insights

Property 1: "Wider is Better" Throughout Training

In μP, wider models consistently achieve better training loss at any point, assuming output layer is zero-initialized.

This contrasts sharply with SP:

Standard Parametrization	Maximal Update Parametrization
· Curves cross frequently	· No curve crossing
· Wider ≠ better reliably	· Wider strictly better
· Scaling unpredictable	· Reliable scaling

Stress Test: GPT-3 Transformer

Scaled width from 256 to 32,768 with fixed HPs:

- Wider models consistently matched or outperformed narrower ones
- Suggests wider models are strictly more data-efficient when scaled appropriately
- Provides cheap way to debug μP implementation: check "wider-is-better" early in training

Property 2: A Theoretical Puzzle

The existence of useful HP transfer reveals a deep theoretical question about neural network training dynamics.

The Paradox:

For useful transfer, we need the HP optimum to converge quickly with width, but the network function should converge slowly.

HP Optimum	Network Function
Converges at small width	Converges very slowly
"Macroscopic" variable	"Microscopic" detail

The Open Question:

Why does this separation exist? Where else should we expect useful HP transfer? Theoretically, it's unclear why HP optimum should converge faster than the function. This remains an open question for future theoretical work.

Practical Implications

1. **Reliable Model Scaling**
   Researchers can confidently scale up models without fear of performance degradation from parametrization issues.
2. **Better Compute Utilization**
   Small model tuning on individual GPUs increases parallelism and better utilizes organizational compute clusters.
3. **Painless Exploration → Scaling**

Future Research Directions

- **Depth Parametrization:** Improve depth transfer, especially for post-LN Transformers
- **Finite-Width Corrections:** Fix slight optimum shifts at smaller scales
- **Regularization Transfer:** Transfer regularization HPs as function of model and data size

CONCLUSION

Impact and Future Directions

Transformative Impact

μTransfer fundamentally changes how we approach hyperparameter tuning for large neural networks, making the previously impossible task of tuning billion-parameter models feasible and efficient.

Massive Speedup

Tuning cost becomes independent of target model size. From 100% to 7% of pretraining cost for GPT-3 6.7B.

Tune Once for All

Single small proxy model provides HPs for entire model families (BERT, GPT, etc.).

Better Performance

μTransferred models outperform SP counterparts even with optimal tuning, due to balanced layer updates.

Better Utilization

Small model tuning on individual GPUs increases parallelism and cluster utilization.

Central Message

μTransfer provides the first major practical payoff of infinite-width theory, transforming HP tuning from an art to a systematic, principled process.

Future Research Directions

1. **Improving Depth Transfer**
   Fix initialization transfer across depth; extend to post-LN Transformers
2. **Finite-Width Corrections**
   Address slight optimum shifts at smaller scales via corrections to μP
3. **Regularization HP Transfer**
   Transfer regularization HPs as function of model and data size
4. **Theoretical Understanding**
   Explain theoretically why HP optimum converges faster than network function

Practical Implementation

PyTorch Package: mup
pip install mup

Documentation:
github.com/microsoft/mup

The μTransfer algorithm and μP parametrization are now accessible to pr