Language Modeling by Language Models

Junyan Cheng, Peter Clark, **Kyle Richardson**

May 2025

Allen Institute for Artificial Intelligence (AI2)

The big picture

- **Fully autonomous discovery**, simulate all aspects of the conventional research process *(e.g., ideation, experiment execution, paper writing)*.

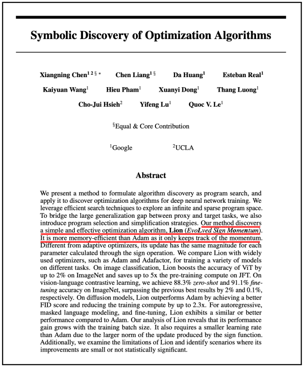
The big picture

- **Autonomous discovery for ML**: Discovering novel machine learning components, make our ML systems more efficient, transparent and safer..

The big picture

- **Autonomous discovery for ML**: Discovering novel machine learning components, make our ML systems more efficient, transparent and safer..

Lion optimizer (Chen et al., 2023)



Symbolic Discovery of Optimization Algorithms

Xiangning Chen[1,2,§,*] Chen Liang[1,§] Da Huang[1] Esteban Real[1]
Kaiyuan Wang[1] Hieu Pham[1] Xuanyi Dong[1] Thang Luong[1]
Cho-Jui Hsieh[2] Yifeng Lu[1] Quoc V. Le[1]

[§]Equal & Core Contribution

[1]Google [2]UCLA

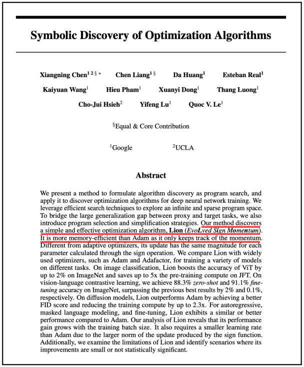
Abstract

We present a method to formulate algorithm discovery as program search, and apply it to discover optimization algorithms for deep neural network training. We leverage efficient search techniques to explore an infinite and sparse program space. To bridge the large generalization gap between proxy and target tasks, we also introduce program selection and simplification strategies. Our method discovers a simple and effective optimization algorithm, **Lion** (*EvoLved Sign Momentum*). It is more memory-efficient than Adam as it only keeps track of the momentum. Different from adaptive optimizers, its update has the same magnitude for each parameter calculated through the sign operation. We compare Lion with widely used optimizers, such as Adam and Adafactor, for training a variety of models on different tasks. On image classification, Lion boosts the accuracy of ViT by up to 2% on ImageNet and saves up to 5x the pre-training compute on JFT. On vision-language contrastive learning, we achieve 88.3% *zero-shot* and 91.1% *fine-tuning* accuracy on ImageNet, surpassing the previous best results by 2% and 0.1%, respectively. On diffusion models, Lion outperforms Adam by achieving a better FID score and reducing the training compute by up to 2.3x. For autoregressive, masked language modeling, and fine-tuning, Lion exhibits a similar or better performance compared to Adam. Our analysis of Lion reveals that its performance gain grows with the training batch size. It also requires a smaller learning rate than Adam due to the larger norm of the update produced by the sign function. Additionally, we examine the limitations of Lion and identify scenarios where its improvements are small or not statistically significant.

The big picture

- **Autonomous discovery for ML**: Discovering novel machine learning components, make our ML systems more efficient, transparent and safer..

Lion optimizer (Chen et al., 2023)



Symbolic Discovery of Optimization Algorithms

Xiangning Chen[1 2 § *] **Chen Liang**[1 §] **Da Huang**[1] **Esteban Real**[1]
Kaiyuan Wang[1] **Hieu Pham**[1] **Xuanyi Dong**[1] **Thang Luong**[1]
Cho-Jui Hsieh[2] **Yifeng Lu**[1] **Quoc V. Le**[1]

[§]Equal & Core Contribution

[1]Google [2]UCLA

Abstract

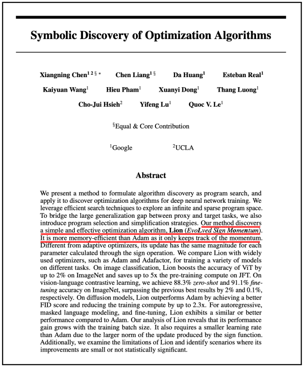
We present a method to formulate algorithm discovery as program search, and apply it to discover optimization algorithms for deep neural network training. We leverage efficient search techniques to explore an infinite and sparse program space. To bridge the large generalization gap between proxy and target tasks, we also introduce program selection and simplification strategies. Our method discovers a simple and effective optimization algorithm, **Lion** (*EvoLved Sign Momentum*). It is more memory-efficient than Adam as it only keeps track of the momentum. Different from adaptive optimizers, its update has the same magnitude for each parameter calculated through the sign operation. We compare Lion with widely used optimizers, such as Adam and Adafactor, for training a variety of models on different tasks. On image classification, Lion boosts the accuracy of ViT by up to 2% on ImageNet and saves up to 5x the pre-training compute on JFT. On vision-language contrastive learning, we achieve 88.3% *zero-shot* and 91.1% *fine-tuning* accuracy on ImageNet, surpassing the previous best results by 2% and 0.1%, respectively. On diffusion models, Lion outperforms Adam by achieving a better FID score and reducing the training compute by up to 2.3x. For autoregressive, masked language modeling, and fine-tuning, Lion exhibits a similar or better performance compared to Adam. Our analysis of Lion reveals that its performance gain grows with the training batch size. It also requires a smaller learning rate than Adam due to the larger norm of the update produced by the sign function. Additionally, we examine the limitations of Lion and identify scenarios where its improvements are small or not statistically significant.

Narrow (no LLMs), clear goals.

The big picture

- **Autonomous discovery for ML**: Discovering novel machine learning components, make our ML systems more efficient, transparent and safer..

Lion optimizer (Chen et al., 2023)



Symbolic Discovery of Optimization Algorithms

Xiangning Chen[1 2 § *] Chen Liang[1 §] Da Huang[1] Esteban Real[1]
Kaiyuan Wang[1] Hieu Pham[1] Xuanyi Dong[1] Thang Luong[1]
Cho-Jui Hsieh[2] Yifeng Lu[1] Quoc V. Le[1]

[§]Equal & Core Contribution

[1]Google [2]UCLA

Abstract

We present a method to formulate algorithm discovery as program search, and apply it to discover optimization algorithms for deep neural network training. We leverage efficient search techniques to explore an infinite and sparse program space. To bridge the large generalization gap between proxy and target tasks, we also introduce program selection and simplification strategies. Our method discovers a simple and effective optimization algorithm, **Lion** (*EvoLved Sign Momentum*). It is more memory-efficient than Adam as it only keeps track of the momentum. Different from adaptive optimizers, its update has the same magnitude for each parameter calculated through the sign operation. We compare Lion with widely used optimizers, such as Adam and Adafactor, for training a variety of models on different tasks. On image classification, Lion boosts the accuracy of ViT by up to 2% on ImageNet and saves up to 5x the pre-training compute on JFT. On vision-language contrastive learning, we achieve 88.3% *zero-shot* and 91.1% *fine-tuning* accuracy on ImageNet, surpassing the previous best results by 2% and 0.1%, respectively. On diffusion models, Lion outperforms Adam by achieving a better FID score and reducing the training compute by up to 2.3x. For autoregressive, masked language modeling, and fine-tuning, Lion exhibits a similar or better performance compared to Adam. Our analysis of Lion reveals that its performance gain grows with the training batch size. It also requires a smaller learning rate than Adam due to the larger norm of the update produced by the sign function. Additionally, we examine the limitations of Lion and identify scenarios where its improvements are small or not statistically significant.

Narrow (no LLMs), clear goals.

AI Scientist (Lu et al., 2024)

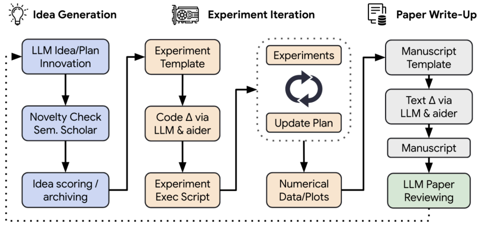
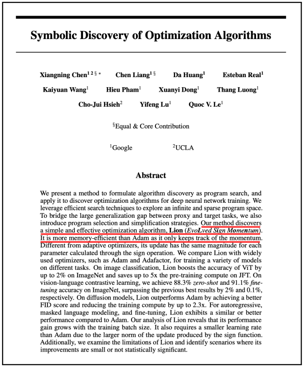


Figure 1 | Conceptual illustration of The AI Scientist, an end-to-end LLM-driven scientific discovery process. The AI Scientist first invents and assesses the novelty of a set of ideas. It then determines how to test the hypotheses, including writing the necessary code by editing a codebase powered by recent advances in automated code generation. Afterward, the experiments are automatically executed to collect a set of results consisting of both numerical scores and visual summaries (e.g. plots or tables). The results are motivated, explained, and summarized in a LaTeX report. Finally, The AI Scientist generates an automated review, according to current practice at standard machine learning conferences. The review can be used to either improve the project or as feedback to future generations for open-ended scientific discovery.

The big picture

- **Autonomous discovery for ML**: Discovering novel machine learning components, make our ML systems more efficient, transparent and safer..

Lion optimizer (Chen et al., 2023)



Symbolic Discovery of Optimization Algorithms

Xiangning Chen[1 2 § *] Chen Liang[1 §] Da Huang[1] Esteban Real[1]
Kaiyuan Wang[1] Hieu Pham[1] Xuanyi Dong[1] Thang Luong[1]
Cho-Jui Hsieh[2] Yifeng Lu[1] Quoc V. Le[1]

[§]Equal & Core Contribution

[1]Google [2]UCLA

Abstract

We present a method to formulate algorithm discovery as program search, and apply it to discover optimization algorithms for deep neural network training. We leverage efficient search techniques to explore an infinite and sparse program space. To bridge the large generalization gap between proxy and target tasks, we also introduce program selection and simplification strategies. Our method discovers a simple and effective optimization algorithm, **Lion** (*EvoLved Sign Momentum*). It is more memory-efficient than Adam as it only keeps track of the momentum. Different from adaptive optimizers, its update has the same magnitude for each parameter calculated through the sign operation. We compare Lion with widely used optimizers, such as Adam and Adafactor, for training a variety of models on different tasks. On image classification, Lion boosts the accuracy of ViT by up to 2% on ImageNet and saves up to 5x the pre-training compute on JFT. On vision-language contrastive learning, we achieve 88.3% *zero-shot* and 91.1% *fine-tuning* accuracy on ImageNet, surpassing the previous best results by 2% and 0.1%, respectively. On diffusion models, Lion outperforms Adam by achieving a better FID score and reducing the training compute by up to 2.3x. For autoregressive, masked language modeling, and fine-tuning, Lion exhibits a similar or better performance compared to Adam. Our analysis of Lion reveals that its performance gain grows with the training batch size. It also requires a smaller learning rate than Adam due to the larger norm of the update produced by the sign function. Additionally, we examine the limitations of Lion and identify scenarios where its improvements are small or not statistically significant.

Narrow (no LLMs), clear goals.

AI Scientist (Lu et al., 2024)

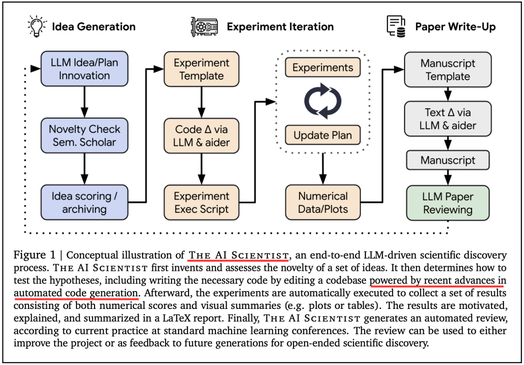
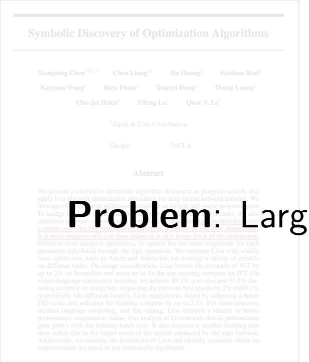


Figure 1 | Conceptual illustration of The AI Scientist, an end-to-end LLM-driven scientific discovery process. The AI Scientist first invents and assesses the novelty of a set of ideas. It then determines how to test the hypotheses, including writing the necessary code by editing a codebase powered by recent advances in automated code generation. Afterward, the experiments are automatically executed to collect a set of results consisting of both numerical scores and visual summaries (e.g. plots or tables). The results are motivated, explained, and summarized in a LaTeX report. Finally, The AI Scientist generates an automated review, according to current practice at standard machine learning conferences. The review can be used to either improve the project or as feedback to future generations for open-ended scientific discovery.

Broad (LLM-driven), unclear goals.

The big picture

Lion optimizer (Chen et al., 2023)



Symbolic Discovery of Optimization Algorithms

Xiangning Chen[1 2 § *] Chen Liang[1 §] Da Huang[1] Esteban Real[1]
Kaiyuan Wang[1] Hieu Pham[1] Xuanyi Dong[1] Thang Luong[1]
Cho-Jui Hsieh[2] Yifeng Lu[1] Quoc V. Le[1]

[§]Equal & Core Contribution

[1]Google [2]UCLA

Abstract

We present a method to formulate algorithm discovery as program search, and apply it to discover optimization algorithms for deep neural network training. We leverage efficient search techniques to explore an infinite and sparse program space. To bridge the large generalization gap between proxy and target tasks, we also introduce program selection and simplification strategies. Our method discovers a simple and effective optimization algorithm, **Lion** (*EvoLved Sign Momentum*). It is more memory-efficient than Adam as it only keeps track of the momentum. Different from adaptive optimizers, its update has the same magnitude for each parameter calculated through the sign operation. We compare Lion with widely used optimizers, such as Adam and Adafactor, for training a variety of models on different tasks. On image classification, Lion boosts the accuracy of ViT by up to 2% on ImageNet and saves up to 5x the pre-training compute on JFT. On vision-language contrastive learning, we achieve 88.3% *zero-shot* and 91.1% *fine-tuning* accuracy on ImageNet, surpassing the previous best results by 2% and 0.1%, respectively. On diffusion models, Lion outperforms Adam by achieving a better FID score and reducing the training compute by up to 2.3x. For autoregressive, masked language modeling, and fine-tuning, Lion exhibits a similar or better performance compared to Adam. Our analysis of Lion reveals that its performance gain grows with the training batch size. It also requires a smaller learning rate than Adam due to the larger norm of the update produced by the sign function. Additionally, we examine the limitations of Lion and identify scenarios where its improvements are small or not statistically significant.

Narrow (no LLMs), clear goals.

AI Scientist (Lu et al., 2024)

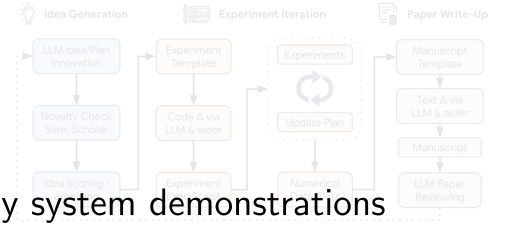


Figure 1 | Conceptual illustration of The AI Scientist, an end-to-end LLM-driven scientific discovery process. The AI Scientist first invents and assesses the novelty of a set of ideas. It then determines how to test the hypotheses, including writing the necessary code by editing a codebase powered by recent advances in automated code generation. Afterward, the experiments are automatically executed to collect a set of results consisting of both numerical scores and visual summaries (e.g. plots or tables). The results are motivated, explained, and summarized in a LaTeX report. Finally, The AI Scientist generates an automated review, according to current practice at standard machine learning conferences. The review can be used to either improve the project or as feedback to future generations for open-ended scientific discovery.

Broad (LLM-driven), unclear goals.

Problem: Largely system demonstrations

1. **Tasks:** What are the target discovery tasks?

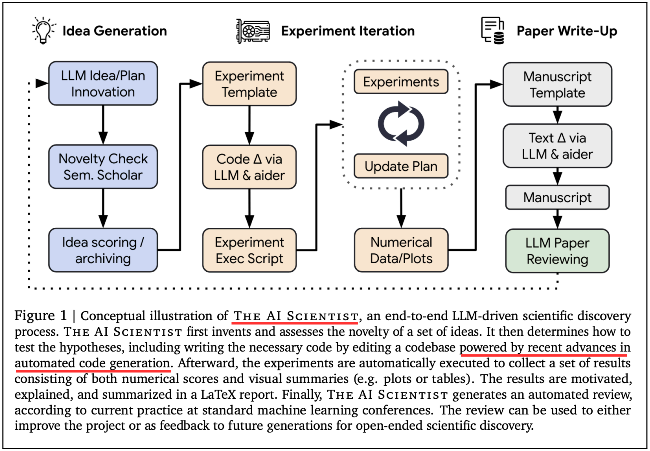
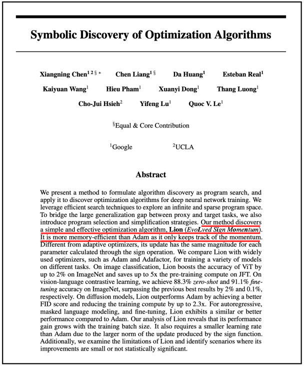


Figure 1 | Conceptual illustration of THE AI SCIENTIST, an end-to-end LLM-driven scientific discovery process. THE AI SCIENTIST first invents and assesses the novelty of a set of ideas. It then determines how to test the hypotheses, including writing the necessary code by editing a codebase powered by recent advances in automated code generation. Afterward, the experiments are automatically executed to collect a set of results consisting of both numerical scores and visual summaries (e.g. plots or tables). The results are motivated, explained, and summarized in a LaTeX report. Finally, THE AI SCIENTIST generates an automated review, according to current practice at standard machine learning conferences. The review can be used to either improve the project or as feedback to future generations for open-ended scientific discovery.

- What tasks and discovery problems should we be working on to make progress? Community has not yet come up with clear tasks or metrics.

1. **Tasks:** What are the target discovery tasks?

Lion optimizer (Chen et al., 2023)



Symbolic Discovery of Optimization Algorithms

Xiangning Chen[1 2 § *] Chen Liang[1 §] Da Huang[1] Esteban Real[1]
Kaiyuan Wang[1] Hieu Pham[1] Xuanyi Dong[1] Thang Luong[1]
Cho-Jui Hsieh[2] Yifeng Lu[1] Quoc V. Le[1]

[§]Equal & Core Contribution

[1]Google [2]UCLA

Abstract

We present a method to formulate algorithm discovery as program search, and apply it to discover optimization algorithms for deep neural network training. We leverage efficient search techniques to explore an infinite and sparse program space. To bridge the large generalization gap between proxy and target tasks, we also introduce program selection and simplification strategies. Our method discovers a simple and effective optimization algorithm, **Lion** (*EvoLved Sign Momentum*). It is more memory-efficient than Adam as it only keeps track of the momentum. Different from adaptive optimizers, its update has the same magnitude for each parameter calculated through the sign operation. We compare Lion with widely used optimizers, such as Adam and Adafactor, for training a variety of models on different tasks. On image classification, Lion boosts the accuracy of ViT by up to 2% on ImageNet and saves up to 5x the pre-training compute on JFT. On vision-language contrastive learning, we achieve 88.3% *zero-shot* and 91.1% *fine-tuning* accuracy on ImageNet, surpassing the previous best results by 2% and 0.1%, respectively. On diffusion models, Lion outperforms Adam by achieving a better FID score and reducing the training compute by up to 2.3x. For autoregressive, masked language modeling, and fine-tuning, Lion exhibits a similar or better performance compared to Adam. Our analysis of Lion reveals that its performance gain grows with the training batch size. It also requires a smaller learning rate than Adam due to the larger norm of the update produced by the sign function. Additionally, we examine the limitations of Lion and identify scenarios where its improvements are small or not statistically significant.

Narrow (no LLMs), clear goals.

AI Scientist (Lu et al., 2024)

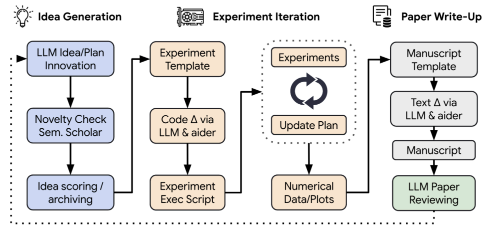
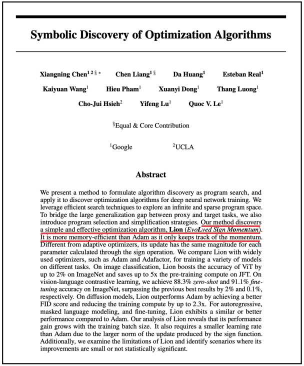


Figure 1 | Conceptual illustration of The AI Scientist, an end-to-end LLM-driven scientific discovery process. The AI Scientist first invents and assesses the novelty of a set of ideas. It then determines how to test the hypotheses, including writing the necessary code by editing a codebase powered by recent advances in automated code generation. Afterward, the experiments are automatically executed to collect a set of results consisting of both numerical scores and visual summaries (e.g. plots or tables). The results are motivated, explained, and summarized in a LaTeX report. Finally, The AI Scientist generates an automated review, according to current practice at standard machine learning conferences. The review can be used to either improve the project or as feedback to future generations for open-ended scientific discovery.

Broad (LLM-driven), unclear goals.

1. **Tasks:** What are the target discovery tasks?

Lion optimizer (Chen et al., 2023)



Symbolic Discovery of Optimization Algorithms

Xiangning Chen[1 2 § *] Chen Liang[1 §] Da Huang[1] Esteban Real[1]
Kaiyuan Wang[1] Hieu Pham[1] Xuanyi Dong[1] Thang Luong[1]
Cho-Jui Hsieh[2] Yifeng Lu[1] Quoc V. Le[1]

[§]Equal & Core Contribution

[1]Google [2]UCLA

Abstract

We present a method to formulate algorithm discovery as program search, and apply it to discover optimization algorithms for deep neural network training. We leverage efficient search techniques to explore an infinite and sparse program space. To bridge the large generalization gap between proxy and target tasks, we also introduce program selection and simplification strategies. Our method discovers a simple and effective optimization algorithm, **Lion** (*EvoLved Sign Momentum*). It is more memory-efficient than Adam as it only keeps track of the momentum. Different from adaptive optimizers, its update has the same magnitude for each parameter calculated through the sign operation. We compare Lion with widely used optimizers, such as Adam and Adafactor, for training a variety of models on different tasks. On image classification, Lion boosts the accuracy of ViT by up to 2% on ImageNet and saves up to 5x the pre-training compute on JFT. On vision-language contrastive learning, we achieve 88.3% *zero-shot* and 91.1% *fine-tuning* accuracy on ImageNet, surpassing the previous best results by 2% and 0.1%, respectively. On diffusion models, Lion outperforms Adam by achieving a better FID score and reducing the training compute by up to 2.3x. For autoregressive, masked language modeling, and fine-tuning, Lion exhibits a similar or better performance compared to Adam. Our analysis of Lion reveals that its performance gain grows with the training batch size. It also requires a smaller learning rate than Adam due to the larger norm of the update produced by the sign function. Additionally, we examine the limitations of Lion and identify scenarios where its improvements are small or not statistically significant.

Narrow (no LLMs), clear goals.

AI Scientist (Lu et al., 2024)

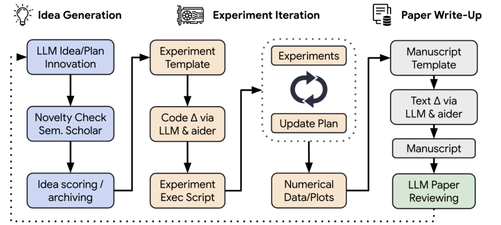


Figure 1 | Conceptual illustration of The AI Scientist, an end-to-end LLM-driven scientific discovery process. The AI Scientist first invents and assesses the novelty of a set of ideas. It then determines how to test the hypotheses, including writing the necessary code by editing a codebase powered by recent advances in automated code generation. Afterward, the experiments are automatically executed to collect a set of results consisting of both numerical scores and visual summaries (e.g. plots or tables). The results are motivated, explained, and summarized in a LaTeX report. Finally, The AI Scientist generates an automated review, according to current practice at standard machine learning conferences. The review can be used to either improve the project or as feedback to future generations for open-ended scientific discovery.

Broad (LLM-driven), unclear goals.

Our proposal: language model architecture discovery, finding better (e.g., more efficient, performant, transparent,...), LM layer designs.

2. **System design**: How should we build such systems?

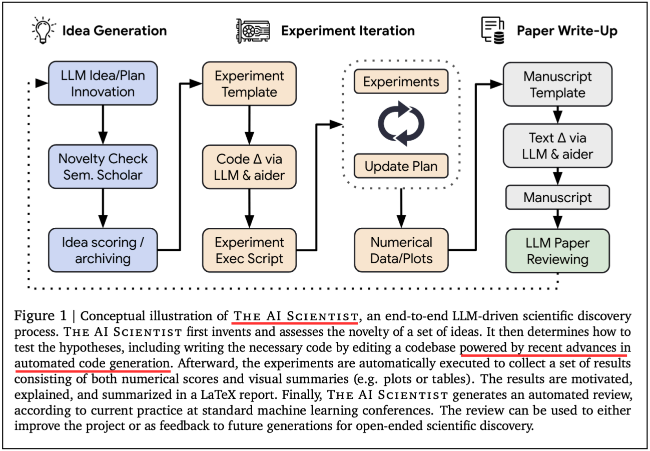


Figure 1 | Conceptual illustration of THE AI SCIENTIST, an end-to-end LLM-driven scientific discovery process. THE AI SCIENTIST first invents and assesses the novelty of a set of ideas. It then determines how to test the hypotheses, including writing the necessary code by editing a codebase powered by recent advances in automated code generation. Afterward, the experiments are automatically executed to collect a set of results consisting of both numerical scores and visual summaries (e.g. plots or tables). The results are motivated, explained, and summarized in a LaTeX report. Finally, THE AI SCIENTIST generates an automated review, according to current practice at standard machine learning conferences. The review can be used to either improve the project or as feedback to future generations for open-ended scientific discovery.

- Do these discovery workflows make sense? What are their limits? Should be efficient, cost-effective, transparent.

2. **System design**: How should we build such systems?

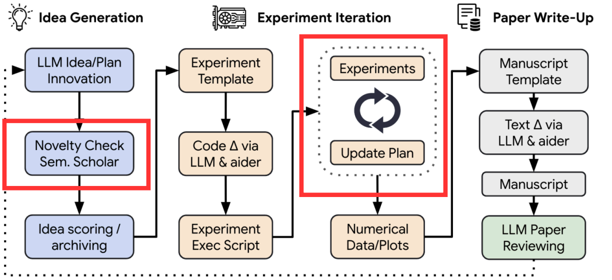


Figure 1 | Conceptual illustration of The AI Scientist, an end-to-end LLM-driven scientific discovery process. The AI Scientist first invents and assesses the novelty of a set of ideas. It then determines how to test the hypotheses, including writing the necessary code by editing a codebase powered by recent advances in automated code generation. Afterward, the experiments are automatically executed to collect a set of results consisting of both numerical scores and visual summaries (e.g. plots or tables). The results are motivated, explained, and summarized in a LaTeX report. Finally, The AI Scientist generates an automated review, according to current practice at standard machine learning conferences. The review can be used to either improve the project or as feedback to future generations for open-ended scientific discovery.

2. **System design**: How should we build such systems?

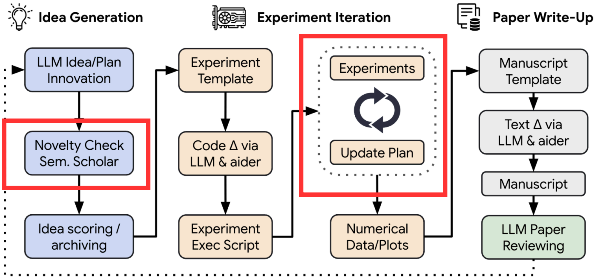
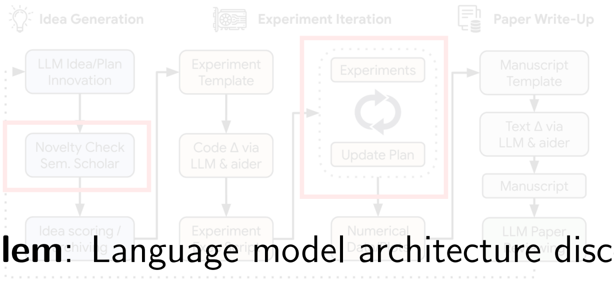


Figure 1 | Conceptual illustration of The AI Scientist, an end-to-end LLM-driven scientific discovery process. The AI Scientist first invents and assesses the novelty of a set of ideas. It then determines how to test the hypotheses, including writing the necessary code by editing a codebase powered by recent advances in automated code generation. Afterward, the experiments are automatically executed to collect a set of results consisting of both numerical scores and visual summaries (e.g. plots or tables). The results are motivated, explained, and summarized in a LaTeX report. Finally, The AI Scientist generates an automated review, according to current practice at standard machine learning conferences. The review can be used to either improve the project or as feedback to future generations for open-ended scientific discovery.

Proposed a new algorithmic framework for discovery, allows us to address technical issues, devise generalized algorithms.

2. **System design**: How should we build such systems?



Problem: Language model architecture discovery

Figure 1 | Conceptual illustration of The AI Scientist, an end-to-end LLM-driven scientific discovery process. The AI Scientist first invents and assesses the novelty of a set of ideas. It then determines how to test the hypotheses, including writing the necessary code by editing a codebase powered by recent advances in automated code generation. Afterward, the experiments are automatically executed to collect a set of results consisting of both numerical scores and visual summaries (e.g. plots or tables). The results are motivated, explained, and summarized in a LaTeX report. Finally, The AI Scientist generates an automated review, according to current practice at standard machine learning conferences. The review can be used to either improve the project or as feedback to future generations for open-ended scientific discovery.

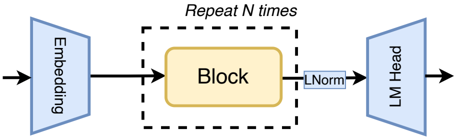
Proposed a new algorithmic framework for discovery, allows us to address technical issues, devise generalized algorithms.

Language model architecture design discovery: what?

- Finding improved layer designs for autoregressive language models.

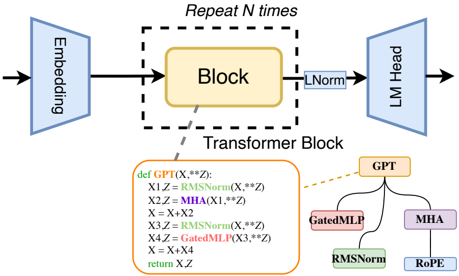
Language model architecture design discovery: what?

- Finding improved layer designs for autoregressive language models.



Language model architecture design discovery: what?

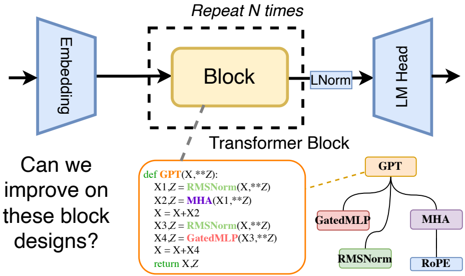
- Finding improved layer designs for autoregressive language models.



```
def GPT(X,**Z):
   X1,Z = RMSNorm(X,**Z)
   X2,Z = MHA(X1,**Z)
   X = X+X2
   X3,Z = RMSNorm(X,**Z)
   X4,Z = GatedMLP(X3,**Z)
   X = X+X4
   return X,Z
```

Language model architecture design discovery: what?

- Finding improved layer designs for autoregressive language models.

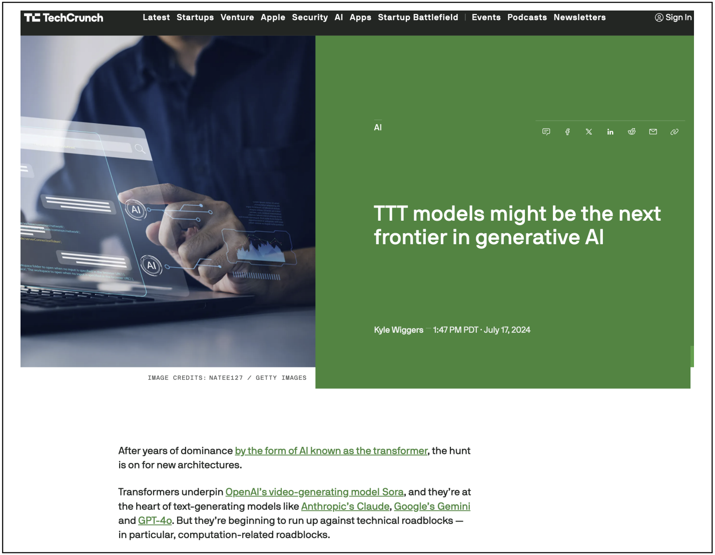


Can we improve on these block designs?

```
def GPT(X,**Z):
   X1,Z = RMSNorm(X,**Z)
   X2,Z = MHA(X1,**Z)
   X = X+X2
   X3,Z = RMSNorm(X,**Z)
   X4,Z = GatedMLP(X3,**Z)
   X = X+X4
   return X,Z
```

At its core, a **code discovery** problem, similar goals to AutoML and Neural architecture search (**NAS**), model full research pipeline.

Why is this an interesting problem?


TechCrunch
Latest Startups Venture Apple Security AI Apps Startup Battlefield | Events Podcasts Newsletters
Sign In
AI

TTT models might be the next frontier in generative AI

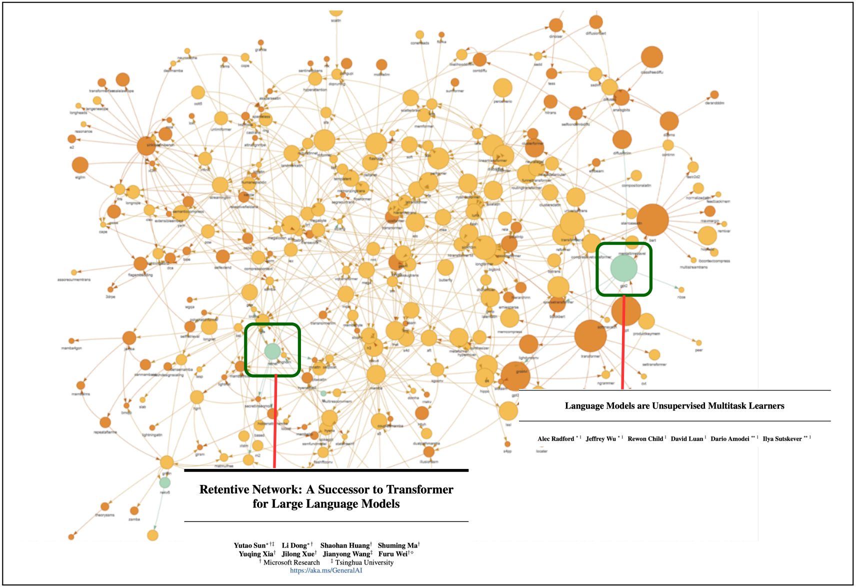
Kyle Wiggers 1:47 PM PDT · July 17, 2024

IMAGE CREDITS: NATEE127 / GETTY IMAGES

After years of dominance by the form of AI known as the transformer, the hunt is on for new architectures.

Transformers underpin OpenAI's video-generating model Sora, and they're at the heart of text-generating models like Anthropic's Claude, Google's Gemini and GPT-4o. But they're beginning to run up against technical roadblocks — in particular, computation-related roadblocks.

Why is this an interesting problem?



Retentive Network: A Successor to Transformer for Large Language Models

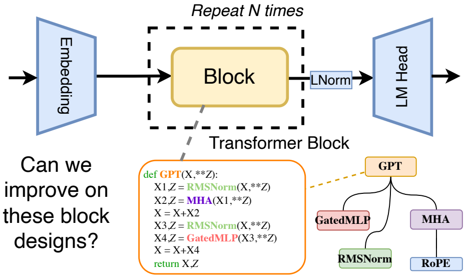
Yutao Sun[*†‡] Li Dong[*†] Shaohan Huang[†] Shuming Ma[†]
Yuqing Xia[†] Jilong Xue[†] Jianyong Wang[‡] Furu Wei[†⋄]
[†] Microsoft Research [‡] Tsinghua University
https://aka.ms/GeneralAI

Language Models are Unsupervised Multitask Learners

Alec Radford[*1] Jeffrey Wu[*1] Rewon Child[1] David Luan[1] Dario Amodei[**1] Ilya Sutskever[**1]

Why is this an interesting problem?

- Finding improved layer designs for autoregressive language models.



```
def GPT(X,**Z):
   X1,Z = RMSNorm(X,**Z)
   X2,Z = MHA(X1,**Z)
   X = X+X2
   X3,Z = RMSNorm(X,**Z)
   X4,Z = GatedMLP(X3,**Z)
   X = X+X4
   return X,Z
```

Can we improve on these block designs?

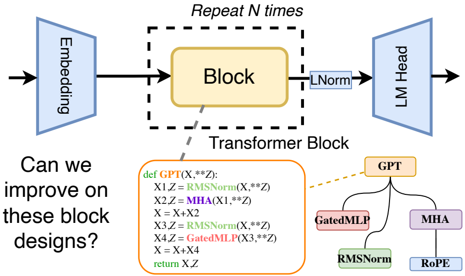
Ill-formed search space: huge unbounded design space.

Complex sampling process: literature understanding, coding skills.

Expensive verification: pre-training/evaluation, resource bound.

Language model architecture design discovery: how?

- Finding improved layer designs for auto-regressive language models.



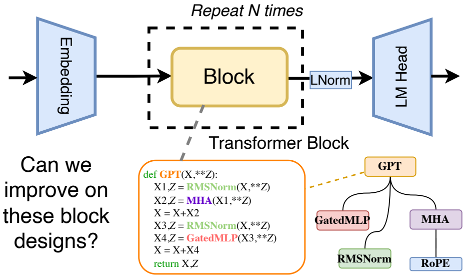
Can we improve on these block designs?

```
def GPT(X,**Z):
   X1,Z = RMSNorm(X,**Z)
   X2,Z = MHA(X1,**Z)
   X = X+X2
   X3,Z = RMSNorm(X,**Z)
   X4,Z = GatedMLP(X3,**Z)
   X = X+X4
   return X,Z
```

Continuous learning loop: Generate new model ideas, implement them and verify through generative pre-training.

Language model architecture design discovery: how?

- Finding improved layer designs for auto-regressive language models.



Can we improve on these block designs?

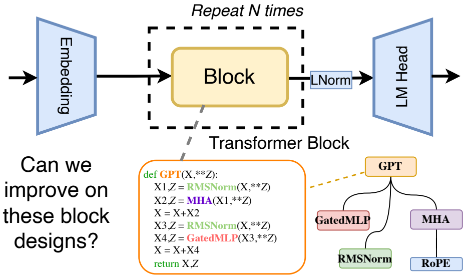
```
def GPT(X,**Z):
   X1,Z = RMSNorm(X,**Z)
   X2,Z = MHA(X1,**Z)
   X = X+X2
   X3,Z = RMSNorm(X,**Z)
   X4,Z = GatedMLP(X3,**Z)
   X = X+X4
   return X,Z
```

Continuous learning loop: Generate new model ideas, implement them and verify through generative pre-training.

- **Objective**: Find designs that improve on end-task performance.

Language model architecture design discovery: how?

- Finding improved layer designs for auto-regressive language models.



```
def GPT(X,**Z):
   X1,Z = RMSNorm(X,**Z)
   X2,Z = MHA(X1,**Z)
   X = X+X2
   X3,Z = RMSNorm(X,**Z)
   X4,Z = GatedMLP(X3,**Z)
   X = X+X4
   return X,Z
```

Can we improve on these block designs?

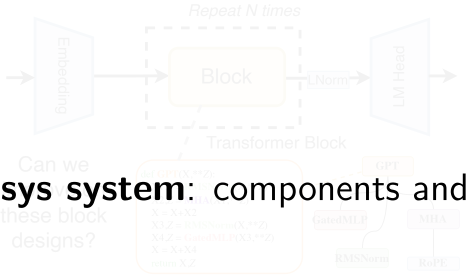
Continuous learning loop: Generate new model ideas, implement them and verify through generative pre-training.

- **Objective**: Find designs that improve on end-task performance.
- Start small, innovate then scale, **Ladder-of-scales** (LoS) approach.

The **Genesys system**: components and principles

Language model architecture design discovery: how?

- Finding improved layer designs for auto-regressive language models.

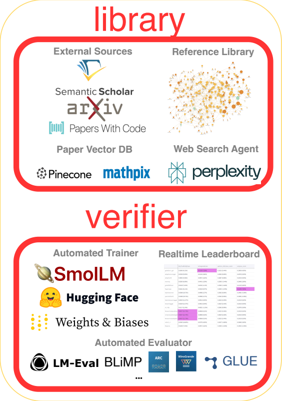


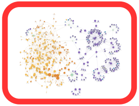
```
def GPT(X,**Z):
  X1,Z = RMSNorm(X,**Z)
  X2,Z = MHA(X1,**Z)
  X = X+X2
  X3,Z = RMSNorm(X,**Z)
  X4,Z = GatedMLP(X3,**Z)
  X = X+X4
  return X,Z
```

Continuous learning loop: Generate new model ideas, implement them and verify through generative pre-training.

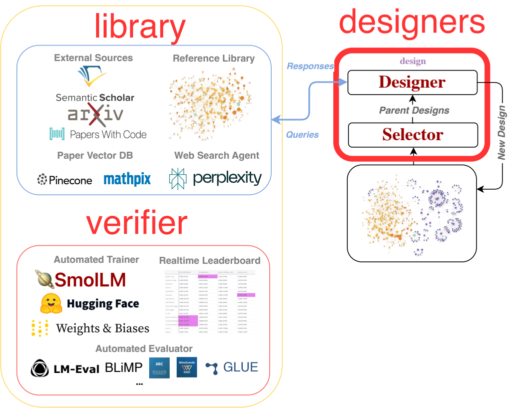
- **Objective**: Find designs that improve on end-task performance.
- Start small, innovate then scale, **Ladder-of-scales** (LoS) approach.

The Genesys system: **core utilities**





The Genesys system: **agents**



library

External Sources

Semantic Scholar

arXiv

Papers With Code

Reference Library

Paper Vector DB

Pinecone mathpix

Web Search Agent

perplexity

verifier

Automated Trainer

SmolLM

Hugging Face

Weights & Biases

Realtime Leaderboard

Automated Evaluator

LM-Eval BLiMP ARC WinoGrande GLUE

...

Responses

Queries

designers

design

Designer

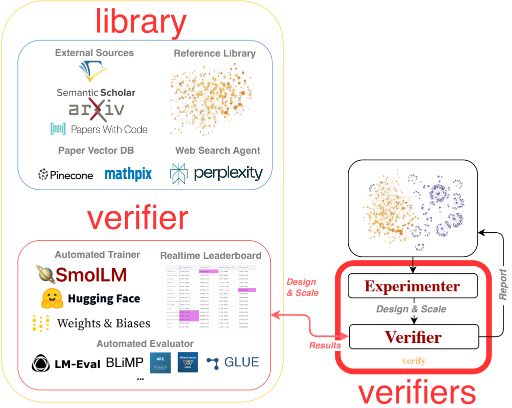
Parent Designs

Selector

New Design

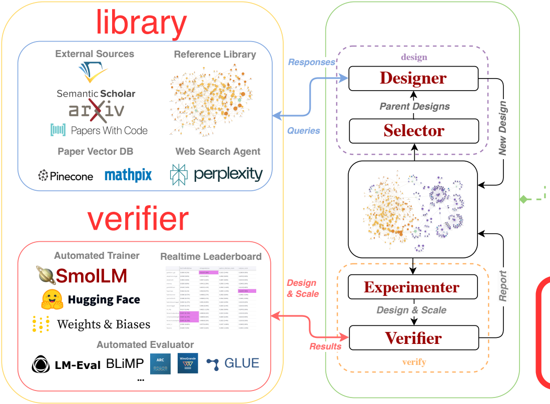
```
async def design(EvoTree, KE):
    """Select parent designs to improve,
    produce new design and add to tree"""
    parents, refs = Selector(EvoTree)
    design = Designer(parents,refs,KE)
    async EvoTree.update(design)
```

The Genesys system: **agents**



```
async def verify(EvoTree, VE):
    """Select a design and scale to verify,
    run experiments and produce a report"""
    design, scale = Experimenter(EvoTree)
    report = Verifier(design,scale,VE)
    async EvoTree.update(design.report)
```

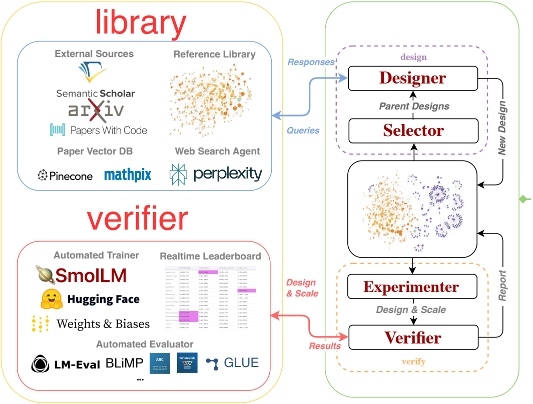
The Genesys system: **distributed evolution**



evolution

```
async def genesys_evolve(EvoTree, KE, VE):
    """Run distributed discovery loop"""
    while budget:
        async design(EvoTree,KE)
        async verify(EvoTree,VE)
```

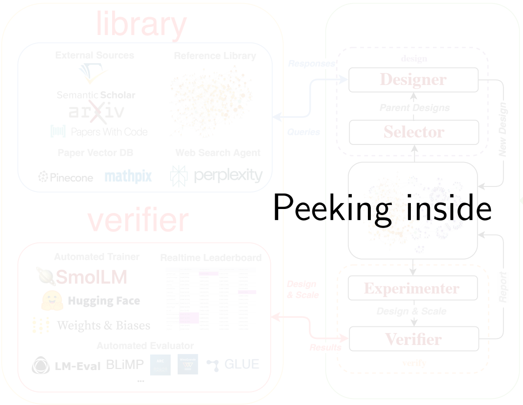
The Genesys system



```
async def design(EvoTree, KE):
    """Select parent designs to improve,
    produce new design and add to tree"""
    parents, refs = Selector(EvoTree)
    design = Designer(parents,refs,KE)
    async EvoTree.update(design)

async def verify(EvoTree, VE):
    """Select a design and scale to verify,
    run experiments and produce a report"""
    design, scale = Experimenter(EvoTree)
    report = Verifier(design,scale,VE)
    async EvoTree.update(design.report)

async def genesys_evolve(EvoTree, KE, VE):
    """Run distributed discovery loop"""
    while budget:
        async design(EvoTree,KE)
        async verify(EvoTree,VE)
```

Experiments at a glance: 1,162 discovered designs (1,062 fully verified), 86K dialogues, 2.76M lines of code, 1B processed tokens.

The Genesys system

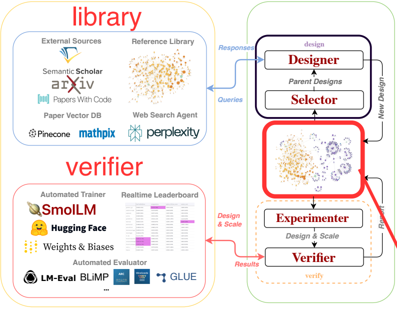


```
async def design(EvoTree, KE):
    """Select parent designs to improve,
    produce new design and add to tree"""
    parents, refs = Selector(EvoTree)
    design = Designer(parents,refs,KE)
    async EvoTree.update(design)

async def verify(EvoTree, VE):
    """Select a design and scale to verify,
    run experiments and produce a report"""
    design, scale = Experimenter(EvoTree)
    report = Verifier(design,scale,VE)
    async EvoTree.update(design,report)

async def genesys_evolve(EvoTree, KE, VE):
    """Run distributed discovery loop"""
    while budget:
        async design(EvoTree,KE)
        async verify(EvoTree,VE)
```

Peeking inside

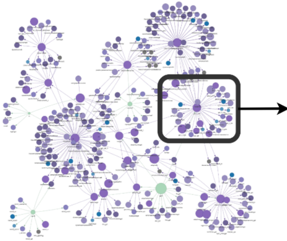
Experiments at a glance: 1,162 discovered designs (1,062 fully verified), 86K dialogues, 2.76M lines of code, 1B processed tokens.

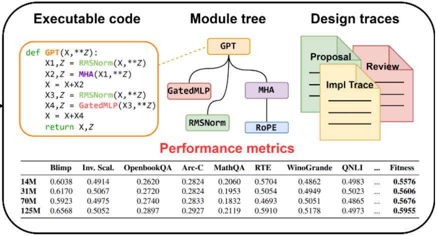
Design tree: fully factorizable design space



Code is fully factorizable, representable as a unit tree

Fitness score: end task performance



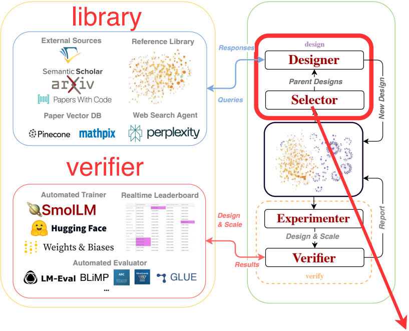


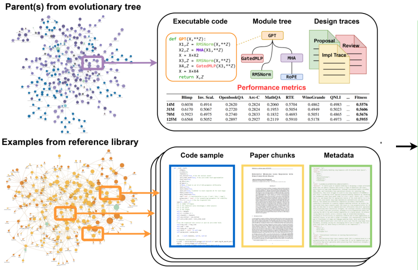
```
def GPT(X,**Z):
    X1,Z = RMSNorm(X,**Z)
    X2,Z = MHA(X1,**Z)
    X = X+X2
    X3,Z = RMSNorm(X,**Z)
    X4,Z = GatedMLP(X3,**Z)
    X = X+X4
    return X,Z
```

Performance metrics

	Blimp	Inv. Scal.	OpenbookQA	Arc-C	MathQA	RTE	WinoGrande	QNLI	...	Fitness
14M	0.6038	0.4914	0.2620	0.2824	0.2060	0.5704	0.4862	0.4983	...	**0.5576**
31M	0.6170	0.5067	0.2720	0.2824	0.1953	0.5054	0.4949	0.5023	...	**0.5606**
70M	0.5923	0.4975	0.2740	0.2833	0.1832	0.4693	0.5051	0.4865	...	**0.5676**
125M	0.6568	0.5052	0.2897	0.2927	0.2119	0.5910	0.5178	0.4973	...	**0.5955**

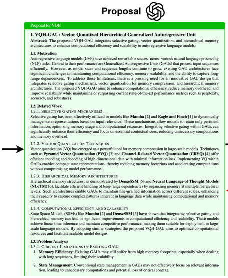
Designers: Proposer-reviewer architecture





```
def GPT(X,**Z):
    X1,Z = RMSNorm(X,**Z)
    X2,Z = MHA(X1,**Z)
    X = X+X2
    X3,Z = RMSNorm(X,**Z)
    X4,Z = GatedMLP(X3,**Z)
    X = X+X4
    return X,Z
```

	Blimp	Inv. Scal.	OpenbookQA	Arc-C	MathQA	RTE	WinoGrande	QNLI	...	Fitness
14M	0.6038	0.4914	0.2620	0.2824	0.2060	0.5704	0.4862	0.4983	...	**0.5576**
31M	0.6170	0.5067	0.2720	0.2824	0.1953	0.5054	0.4949	0.5023	...	**0.5606**
70M	0.5923	0.4975	0.2740	0.2833	0.1832	0.4693	0.5051	0.4865	...	**0.5676**
125M	0.6568	0.5052	0.2897	0.2927	0.2119	0.5910	0.5178	0.4973	...	**0.5955**



Proposal

Proposal for VQH

I. VQH-GAU: Vector Quantized Hierarchical Generalized Autoregressive Unit

Abstract: The proposed VQH-GAU integrates selective gating, vector quantization, and hierarchical memory architectures to enhance computational efficiency and scalability in autoregressive language models.

I.1. Motivation

Autoregressive language models (LMs) have achieved remarkable success across various natural language processing (NLP) tasks. Central to their performance are Generalized Autoregressive Units (GAUs) that process input sequences efficiently. However, as model sizes and sequence lengths continue to grow, existing GAU architectures face significant challenges in maintaining computational efficiency, memory scalability, and the ability to capture long-range dependencies. To address these limitations, there is a pressing need for an innovative GAU design that integrates selective gating mechanisms, vector quantization for memory compression, and hierarchical memory architectures. The proposed VQH-GAU aims to enhance computational efficiency, reduce memory overhead, and improve scalability while maintaining or surpassing current state-of-the-art performance metrics such as perplexity, accuracy, and robustness.

I.2. Related Work

I.2.1. Selective Gating Mechanisms

Selective gating has been effectively utilized in models like **Mamba** [2] and **Eagle and Finch** [1] to dynamically manage state representations based on input relevance. These mechanisms allow models to retain only pertinent information, optimizing memory usage and computational resources. Integrating selective gating within GAUs can significantly enhance their efficiency and focus on essential contextual cues, reducing unnecessary computations and memory overhead.

I.2.2. Vector Quantization Techniques

Vector quantization (VQ) has emerged as a powerful tool for memory compression in large-scale models. Techniques such as **Pyramid Vector Quantization (PVQ)** [3] and **Channel-Relaxed Vector Quantization (CRVQ)** [4] offer efficient encoding and decoding of high-dimensional data with minimal information loss. Implementing VQ within GAUs enables compact state representations, thereby reducing memory footprints and accelerating computations without compromising model performance.

I.2.3. Hierarchical Memory Architectures

Hierarchical memory structures, as demonstrated by **DenseSSM** [5] and **Neural Language of Thought Models (NLoTM)** [6], facilitate efficient handling of long-range dependencies by organizing memory at multiple hierarchical levels. Such architectures enable GAUs to maintain fine-grained information across different scales, enhancing their capacity to capture complex patterns inherent in language data while maintaining computational and memory efficiency.

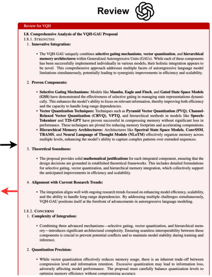
I.2.4. Computational Efficiency and Scalability

State Space Models (SSMs) like **Mamba** [2] and **DenseSSM** [5] have shown that integrating selective gating and hierarchical memory can lead to significant improvements in computational efficiency and scalability. These models achieve linear-time inference and maintain competitive performance, making them suitable for deployment in large-scale language models. By adopting similar strategies, the proposed VQH-GAU aims to optimize computational resources and facilitate scalable model designs.

I.3. Problem Analysis

I.3.1. Current Limitations of Existing GAUs

1. **Memory Efficiency**: Existing GAUs may still suffer from high memory footprints, especially when dealing with long sequences, limiting their scalability.
2. **State Management**: Conventional state management in GAUs may not effectively focus on relevant information, leading to unnecessary computations and potential loss of critical context.



Review

Review for VQH

I.8. Comprehensive Analysis of the VQH-GAU Proposal

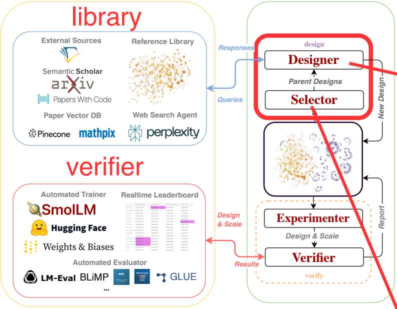
I.8.1. Strengths

1. **Innovative Integration:**
   - The VQH-GAU uniquely combines **selective gating mechanisms**, **vector quantization**, and **hierarchical memory architectures** within Generalized Autoregressive Units (GAUs). While each of these components has been successfully implemented individually in various models, their holistic integration appears to be novel. This comprehensive approach addresses multiple facets of autoregressive language model limitations simultaneously, potentially leading to synergistic improvements in efficiency and scalability.
2. **Proven Components:**
   - **Selective Gating Mechanisms:** Models like **Mamba**, **Eagle and Finch**, and **Gated State Space Models (GSS)** have demonstrated the effectiveness of selective gating in managing state representations dynamically. This enhances the model's ability to focus on relevant information, thereby improving both efficiency and the capacity to handle long-range dependencies.
   - **Vector Quantization Techniques:** Techniques such as **Pyramid Vector Quantization (PVQ)**, **Channel-Relaxed Vector Quantization (CRVQ)**, **VPTQ**, and hierarchical methods in models like **Speech-Tokenizer** and **T2S-GPT** have proven successful in compressing memory without significant loss in performance. These techniques are pivotal for reducing memory footprints and accelerating computations.
   - **Hierarchical Memory Architectures:** Architectures like **Spectral State Space Models**, **ConvSSM**, **TRAMS**, and **Neural Language of Thought Models (NLoTM)** effectively organize memory across multiple levels, enhancing the model's ability to capture complex patterns over extended sequences.
3. **Theoretical Soundness:**
   - The proposal provides solid **mathematical justifications** for each integrated component, ensuring that the design decisions are grounded in established theoretical frameworks. This includes detailed formulations for selective gating, vector quantization, and hierarchical memory integration, which collectively support the anticipated improvements in efficiency and scalability.
4. **Alignment with Current Research Trends:**
   - The integration aligns well with ongoing research trends focused on enhancing model efficiency, scalability, and the ability to handle long-range dependencies. By addressing multiple challenges simultaneously, VQH-GAU positions itself at the forefront of advancements in autoregressive language modeling.

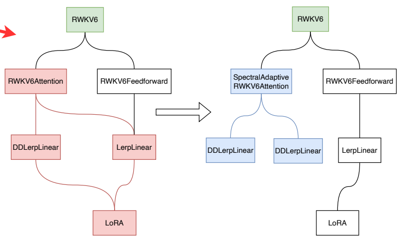
I.8.2. Concerns

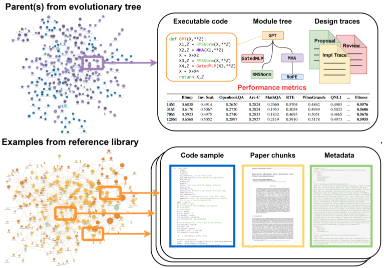
1. **Complexity of Integration:**
   - Combining three advanced mechanisms—selective gating, vector quantization, and hierarchical memory—introduces significant architectural complexity. Ensuring seamless interoperability between these components is crucial to prevent potential conflicts and to maintain model stability during training and inference.
2. **Quantization Precision:**
   - While vector quantization effectively reduces memory usage, there is an inherent trade-off between compression level and information retention. Excessive quantization may lead to information loss, adversely affecting model performance. The proposal must carefully balance quantization levels to optimize memory efficiency without compromising accuracy.

Designers: Planner, coder, observer



Unit-based design and implementation





Parent(s) from evolutionary tree

Executable code

```
def GPT(X,**Z):
    X1,Z = RMSNorm(X,**Z)
    X2,Z = MHA(X1,**Z)
    X = X+X2
    X3,Z = RMSNorm(X,**Z)
    X4,Z = GatedMLP(X3,**Z)
    X = X+X4
    return X,Z
```

Module tree

Design traces

Performance metrics

	Blimp	Inv. Scal.	OpenbookQA	Arc-C	MathQA	RTE	WinoGrande	QNLI	...	Fitness
14M	0.6038	0.4914	0.2620	0.2824	0.2060	0.5704	0.4862	0.4983	...	**0.5576**
31M	0.6170	0.5067	0.2720	0.2824	0.1953	0.5054	0.4949	0.5023	...	**0.5606**
70M	0.5923	0.4975	0.2740	0.2833	0.1832	0.4693	0.5051	0.4865	...	**0.5676**
125M	0.6568	0.5052	0.2897	0.2927	0.2119	0.5910	0.5178	0.4973	...	**0.5955**

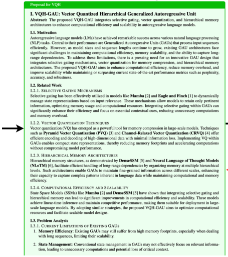
Examples from reference library

Code sample

Paper chunks

Metadata

Proposal



Proposal for VQH

I. VQH-GAU: Vector Quantized Hierarchical Generalized Autoregressive Unit

Abstract: The proposed VQH-GAU integrates selective gating, vector quantization, and hierarchical memory architectures to enhance computational efficiency and scalability in autoregressive language models.

I.1. Motivation

Autoregressive language models (LMs) have achieved remarkable success across various natural language processing (NLP) tasks. Central to their performance are Generalized Autoregressive Units (GAUs) that process input sequences efficiently. However, as model sizes and sequence lengths continue to grow, existing GAU architectures face significant challenges in maintaining computational efficiency, memory scalability, and the ability to capture long-range dependencies. To address these limitations, there is a pressing need for an innovative GAU design that integrates selective gating mechanisms, vector quantization for memory compression, and hierarchical memory architectures. The proposed VQH-GAU aims to enhance computational efficiency, reduce memory overhead, and improve scalability while maintaining or surpassing current state-of-the-art performance metrics such as perplexity, accuracy, and robustness.

I.2. Related Work

I.2.1. Selective Gating Mechanisms

Selective gating has been effectively utilized in models like **Mamba** [2] and **Eagle and Finch** [1] to dynamically manage state representations based on input relevance. These mechanisms allow models to retain only pertinent information, optimizing memory usage and computational resources. Integrating selective gating within GAUs can significantly enhance their efficiency and focus on essential contextual cues, reducing unnecessary computations and memory overhead.

I.2.2. Vector Quantization Techniques

Vector quantization (VQ) has emerged as a powerful tool for memory compression in large-scale models. Techniques such as **Pyramid Vector Quantization (PVQ)** [3] and **Channel-Relaxed Vector Quantization (CRVQ)** [4] offer efficient encoding and decoding of high-dimensional data with minimal information loss. Implementing VQ within GAUs enables compact state representations, thereby reducing memory footprints and accelerating computations without compromising model performance.

I.2.3. Hierarchical Memory Architectures

Hierarchical memory structures, as demonstrated by **DenseSSM** [5] and **Neural Language of Thought Models (NLoTM)** [6], facilitate efficient handling of long-range dependencies by organizing memory at multiple hierarchical levels. Such architectures enable GAUs to maintain fine-grained information across different scales, enhancing their capacity to capture complex patterns inherent in language data while maintaining computational and memory efficiency.

I.2.4. Computational Efficiency and Scalability

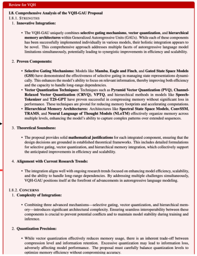
State Space Models (SSMs) like **Mamba** [2] and **DenseSSM** [5] have shown that integrating selective gating and hierarchical memory can lead to significant improvements in computational efficiency and scalability. These models achieve linear-time inference and maintain competitive performance, making them suitable for deployment in large-scale language models. By adopting similar strategies, the proposed VQH-GAU aims to optimize computational resources and facilitate scalable model designs.

I.3. Problem Analysis

I.3.1. Current Limitations of Existing GAUs

1. **Memory Efficiency**: Existing GAUs may still suffer from high memory footprints, especially when dealing with long sequences, limiting their scalability.
2. **State Management**: Conventional state management in GAUs may not effectively focus on relevant information, leading to unnecessary computations and potential loss of critical context.

Review



Review for VQH

I.8. Comprehensive Analysis of the VQH-GAU Proposal

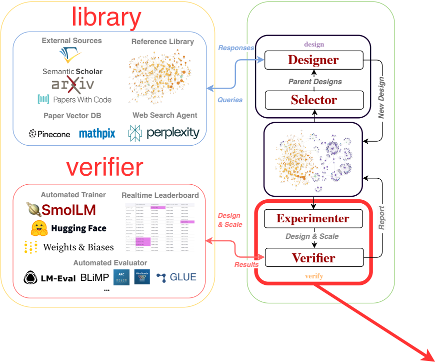
I.8.1. Strengths

1. **Innovative Integration:**
   - The VQH-GAU uniquely combines **selective gating mechanisms**, **vector quantization**, and **hierarchical memory architectures** within Generalized Autoregressive Units (GAUs). While each of these components has been successfully implemented individually in various models, their holistic integration appears to be novel. This comprehensive approach addresses multiple facets of autoregressive language model limitations simultaneously, potentially leading to synergistic improvements in efficiency and scalability.
2. **Proven Components:**
   - **Selective Gating Mechanisms:** Models like **Mamba**, **Eagle and Finch**, and **Gated State Space Models (GSS)** have demonstrated the effectiveness of selective gating in managing state representations dynamically. This enhances the model's ability to focus on relevant information, thereby improving both efficiency and the capacity to handle long-range dependencies.
   - **Vector Quantization Techniques:** Techniques such as **Pyramid Vector Quantization (PVQ)**, **Channel-Relaxed Vector Quantization (CRVQ)**, **VPTQ**, and hierarchical methods in models like **Speech-Tokenizer** and **T2S-GPT** have proven successful in compressing memory without significant loss in performance. These techniques are pivotal for reducing memory footprints and accelerating computations.
   - **Hierarchical Memory Architectures:** Architectures like **Spectral State Space Models**, **ConvSSM**, **TRAMS**, and **Neural Language of Thought Models (NLoTM)** effectively organize memory across multiple levels, enhancing the model's ability to capture complex patterns over extended sequences.
3. **Theoretical Soundness:**
   - The proposal provides solid **mathematical justifications** for each integrated component, ensuring that the design decisions are grounded in established theoretical frameworks. This includes detailed formulations for selective gating, vector quantization, and hierarchical memory integration, which collectively support the anticipated improvements in efficiency and scalability.
4. **Alignment with Current Research Trends:**
   - The integration aligns well with ongoing research trends focused on enhancing model efficiency, scalability, and the ability to handle long-range dependencies. By addressing multiple challenges simultaneously, VQH-GAU positions itself at the forefront of advancements in autoregressive language modeling.

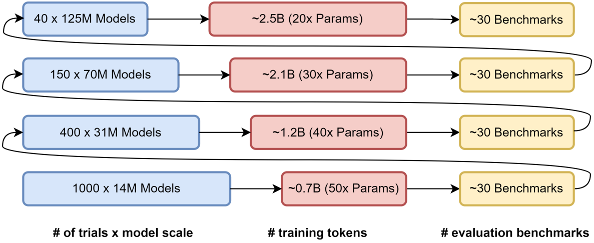
I.8.2. Concerns

1. **Complexity of Integration:**
   - Combining three advanced mechanisms—selective gating, vector quantization, and hierarchical memory—introduces significant architectural complexity. Ensuring seamless interoperability between these components is crucial to prevent potential conflicts and to maintain model stability during training and inference.
2. **Quantization Precision:**
   - While vector quantization effectively reduces memory usage, there is an inherent trade-off between compression level and information retention. Excessive quantization may lead to information loss, adversely affecting model performance. The proposal must carefully balance quantization levels to optimize memory efficiency without compromising accuracy.

Verifiers: budget sensitive scaling



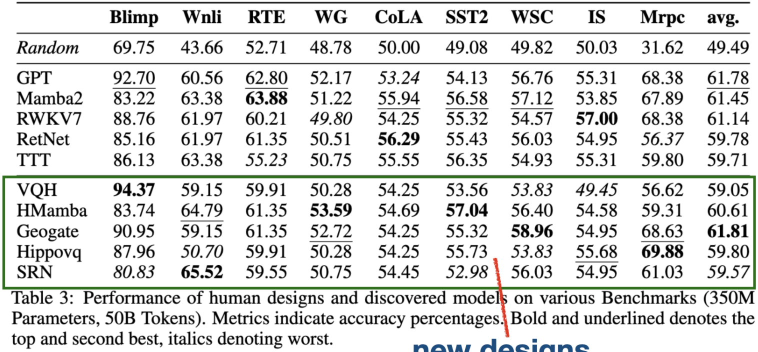
Ladder of scales (LoS) approach



A sketch of the results: **end task performance**

Have we made any discoveries yet?

A sketch of the results: **end task performance**

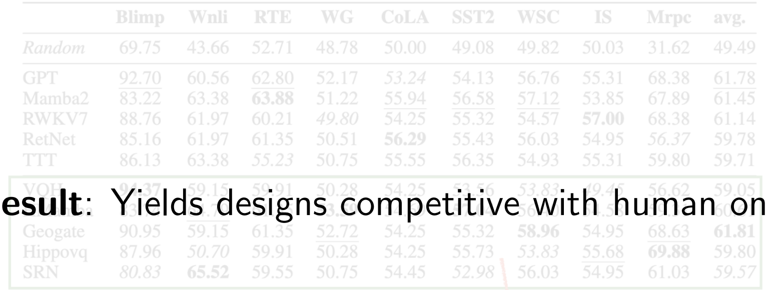


	Blimp	Wnli	RTE	WG	CoLA	SST2	WSC	IS	Mrpc	avg.
Random	69.75	43.66	52.71	48.78	50.00	49.08	49.82	50.03	31.62	49.49
GPT	92.70	60.56	62.80	52.17	*53.24*	54.13	56.76	55.31	68.38	61.78
Mamba2	83.22	63.38	**63.88**	51.22	55.94	56.58	57.12	53.85	67.89	61.45
RWKV7	88.76	61.97	60.21	*49.80*	54.25	55.32	54.57	**57.00**	68.38	61.14
RetNet	85.16	61.97	61.35	50.51	**56.29**	55.43	56.03	54.95	*56.37*	59.78
TTT	86.13	63.38	*55.23*	50.75	55.55	56.35	54.93	55.31	59.80	59.71
VQH	**94.37**	59.15	59.91	50.28	54.25	53.56	*53.83*	*49.45*	56.62	59.05
HMamba	83.74	64.79	61.35	**53.59**	54.69	**57.04**	56.40	54.58	59.31	60.61
Geogate	90.95	59.15	61.35	52.72	54.25	55.32	**58.96**	54.95	68.63	**61.81**
Hippovq	87.96	*50.70*	59.91	50.28	54.25	55.73	*53.83*	55.68	**69.88**	59.80
SRN	*80.83*	**65.52**	59.55	50.75	54.45	*52.98*	56.03	54.95	61.03	*59.57*

Table 3: Performance of human designs and discovered models on various Benchmarks (350M Parameters, 50B Tokens). Metrics indicate accuracy percentages. Bold and underlined denotes the top and second best, italics denoting worst.

new designs

A sketch of the results: **end task performance**



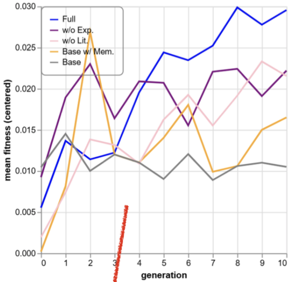
	Blimp	Wnli	RTE	WG	CoLA	SST2	WSC	IS	Mrpc	avg.
Random	69.75	43.66	52.71	48.78	50.00	49.08	49.82	50.03	31.62	49.49
GPT	92.70	60.56	62.80	52.17	*53.24*	54.13	56.76	55.31	68.38	61.78
Mamba2	83.22	63.38	**63.88**	51.22	55.94	56.58	57.12	53.85	67.89	61.45
RWKV7	88.76	61.97	60.21	*49.80*	54.25	55.32	54.57	**57.00**	68.38	61.14
RetNet	85.16	61.97	61.35	50.51	**56.29**	55.43	56.03	54.95	*56.37*	59.78
TTT	86.13	63.38	*55.23*	50.75	55.55	56.35	54.93	55.31	59.80	59.71
VQH	**94.37**	59.15	59.91	50.28	54.25	53.56	*53.83*	*49.45*	56.62	59.05
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Geogate	90.95	59.15	61.35	52.72	54.25	55.32	**58.96**	54.95	68.63	**61.81**
Hippovq	87.96	*50.70*	59.91	50.28	54.25	55.73	*53.83*	55.68	**69.88**	59.80
SRN	*80.83*	**65.52**	59.55	50.75	54.45	*52.98*	56.03	54.95	61.03	*59.57*

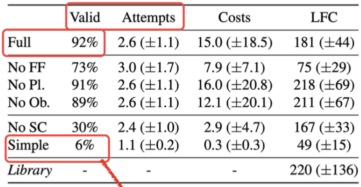
Table 3: Performance of human designs and discovered models on various Benchmarks (350M Parameters, 50B Tokens). Metrics indicate accuracy percentages. Bold and underlined denotes the top and second best, italics denoting worst.

new designs

Result: Yields designs competitive with human ones

A sketch of the results: **system and design analysis**





	Valid	Attempts	Costs	LFC
Full	92%	2.6 (±1.1)	15.0 (±18.5)	181 (±44)
No FF	73%	3.0 (±1.7)	7.9 (±7.1)	75 (±29)
No Pl.	91%	2.6 (±1.1)	16.0 (±20.8)	218 (±69)
No Ob.	89%	2.6 (±1.1)	12.1 (±20.1)	211 (±67)
No SC	30%	2.4 (±1.0)	2.9 (±4.7)	167 (±33)
Simple	6%	1.1 (±0.2)	0.3 (±0.3)	49 (±15)
Library	-	-	-	220 (±136)

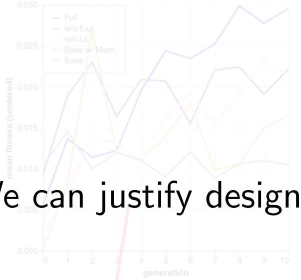
Table 3. Agent benchmark results. Bold and underlined denotes the top and second best. "Library" stands for our reference library with 180 designs providing core block code.

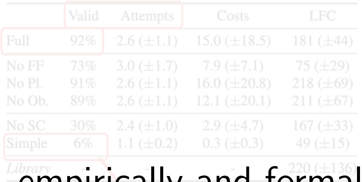
system stability

successful code generation rates

A sketch of the results: **system and design analysis**

We can justify design, empirically and formally.





	Valid	Attempts	Costs	LFC
Full	92%	2.6 (±1.1)	15.0 (±18.5)	181 (±44)
No FF	73%	3.0 (±1.7)	7.9 (±7.1)	75 (±29)
No Pl.	91%	2.6 (±1.1)	16.0 (±20.8)	218 (±69)
No Ob.	89%	2.6 (±1.1)	12.1 (±20.1)	211 (±67)
No SC	30%	2.4 (±1.0)	2.9 (±4.7)	167 (±33)
Simple	6%	1.1 (±0.2)	0.3 (±0.3)	49 (±15)
Library	-	-	-	220 (±136)

Table 3. Agent benchmark results. Bold and underlined denotes the top and second best. “Library” stands for our reference library with 180 designs providing core block code.

successful code generation rates

system stability

Please come to the poster to learn more

Thank you.

References I

Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Pham, H., Dong, X., Luong, T., Hsieh, C.-J., Lu, Y., et al. (2023). Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36:49205–49233.

Lu, C., Lu, C., Lange, R. T., Foerster, J., Clune, J., and Ha, D. (2024). The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*.