Contents

CEP-EIT-P

: CEP-EIT-P
:
: 2-4

---

1 CEP

1-2 - Einstein - -

3-4 CEP - $E = mc^2 + \Delta EF + \Delta ES + \cdot EC$ - - CEP

5 - CEP DOI: 10.5281/zenodo.17301897 - CEP - AGI

---

2 EIT-P

1-2 - IEM Intelligence Emergence Mechanism - EIT-P -

3-4 - - -

5 - EIT-P DOI: 10.5281/zenodo.17298818 - simple_demo.py -

3

1-2 - - GPU -

3-4 - EIT-P - CEP -

5 - - - A/B

4

1-2 - - consciousness_detection_tool -

3-4 - CEP - -

5 - - CEP -

---

1 CEP

-
- CEP
-

1.1 Einstein

$E = mc^2$

- - -

- - -

1.2 CEP

$E = mc^2 + \Delta EF + \Delta ES + \cdot EC$

- mc^2 -
- ΔEF -
- ΔES -
- $\cdot EC$ -

1.3

mc^2 ( ):

-
-

ΔEF ( ):

-
-

ΔES ( ):

-
- Landauer kT·ln(2) bit

·EC ( ):

-
-
-
- EC = k·D·TC ×

1.4

CEP

```
1.    : D   2.7
2.     :    0.8
3.    : Ω   0 (  )
```

1.1: CEP

```
#   10 256
#   CEP

import torch
import torch.nn as nn

model = nn.Sequential(*[nn.Linear(256, 256) for _ in range(10)])

#
# 1.       D
# 2.
# 3.

#     simple_cep_validation_test.py
```

1.2:

1. AI 2. “ ” 3.

1.3:

1. CEP https://doi.org/10.5281/zenodo.17301897 2. Abstract Introduction 3. CEP

---

2 EIT-P

- EIT-P CEP
- IEM
-

2.1 IEM

Intelligence Emergence Mechanism:

```
IEM =  · H · T · C
```

- H - Information Entropy
- T - Temperature
- C - Coherence
- -

```
H ( ):
T ( ):
C (  ):
```

2.2

1:

```
Landauer
```

- = kT·ln(2) per bit
-
- thermodynamic_loss.py

2:

- →
- →
- →
- chaos.py

3:

-
-
-
- path_norm.py

2.3 EIT-P

```
Input Data
```

EIT-P Transformer

- Self-Attention ( )
- Feed-Forward ( )
- Layer Norm ( )

CEP Parameter Monitoring

- D
-
- Ω
- IEM

Output + Consciousness Metrics

2.1:

```
#  EIT-P
cd /mnt/sda1/myproject/datainall/AGI_clean

#
# 1. eit_p/training/eitp_trainer.py
# 2. eit_p/losses/thermodynamic_loss.py
# 3. eit_p/regularization/chaos.py

#
```

2.2: Demo

```
#   demo
python simple_demo.py

#
# •  loss
# • CEP
# •
```

2.3:

simple_demo.py

```
#    alpha
alpha_values = [0.1, 0.5, 1.0, 2.0]

#
```

---

3 -

- EIT-P
-
-

3.1

```
#  GitHub
git clone https://github.com/f21211/eitp-real-product.git
cd eitp-real-product

#
ls -la
```

3.2

```
#  Python
python3 -m venv venv

#
source venv/bin/activate  # Linux/Mac
#
venv\Scripts\activate  # Windows
```

3.3

```
#
pip install -r requirements.txt

#
python -c "import torch; print(f'PyTorch {torch.__version__}')"
python -c "import transformers; print(f'Transformers installed')"
```

3.4

```
#  GPU
python -c "import torch; print(f'CUDA available: {torch.cuda.is_available()}')"
```

```
#   GPU
python -c "import torch; print(f'GPU: {torch.cuda.get_device_name(0)}')"
```

3.5 Demo

```
#    demo
python simple_demo.py

#
# •
# • CEP
# •
```

3.1:

- [ ] Python 3.9+ - [ ] PyTorch 2.0+ - [ ] Transformers - [ ] GPU

3.2:

config.yaml

```
#
model:
  layers: 4   #   6   8
  dim: 256    #   512

training:
  epochs: 10   #   5
```

4

- EIT-P
-
- CEP

4.1

```
#
#

#
echo "This is a sample text for training." > data.txt
```

4.2

my_first_training.py

```
#!/usr/bin/env python3
"""
   EIT-P
"""

import torch
from eit_p.training.eitp_trainer import EITPTrainer
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# 1.    tokenizer
print("  ...")
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')

# 2.
print("  ...")
train_texts = [
    "The quick brown fox jumps over the lazy dog.",
    "Machine learning is transforming the world.",
    "Physics provides the foundation for intelligence.",
]

# Tokenize
train_encodings = tokenizer(train_texts, truncation=True,
                            padding=True, return_tensors='pt')

# 3.  EIT-P
print(" EIT-P  ...")
trainer = EITPTrainer(
    model=model,
    alpha=1.0,  # IEM
    enable_thermodynamic=True,
    enable_emergence=True,
    enable_complexity=True
)

# 4.
print("  ...")
trainer.train(
    train_data=train_encodings,
    epochs=5,
    batch_size=2,
    learning_rate=5e-5
)
```

```
# 5.  CEP
print("\nCEP :")
print(f"   D: {trainer.fractal_dimension:.3f}")
print(f"    : {trainer.complexity_coefficient:.3f}")
print(f"IEM : {trainer.iem_energy:.6f}")

# 6.
print("\n  :")
test_input = tokenizer("The future of AI is", return_tensors='pt')
output = model.generate(**test_input, max_length=20)
generated_text = tokenizer.decode(output[0])
print(generated_text)

print("\n    ")
```

4.3

```
python my_first_training.py
```

4.4

```
Epoch 1/5:
  Loss: 3.456
  IEM Energy: 0.0234
  Fractal Dimension: 2.45
  Complexity Coefficient: 0.67

→    D
→ D 2.7,  0.8
```

4.1:

- Loss - CEP -

4.2: alpha

alpha = [0.1, 0.5, 1.0, 2.0, 5.0] - alpha - alpha

4.3:

```
#
energy_history = []
for epoch in range(epochs):
    energy = trainer.calculate_total_energy()
    energy_history.append(energy)

#
import matplotlib.pyplot as plt
plt.plot(energy_history)
```

```
plt.xlabel('Epoch')
plt.ylabel('Total Energy')
plt.title('CEP Energy Evolution')
plt.savefig('energy_curve.png')
```

5

-
- consciousness_detection_tool
-

5.1

CEP

```
Consciousness Level (0-10) = f(D,  , Ω, H, C)
```

- D -
- -
- Ω -
- H -
- C -

5.2

```
from consciousness_detection_tool import ConsciousnessDetector

#
detector = ConsciousnessDetector()

#
test_systems = {
    '  ': torch.randn(32, 64),
    '  ': load_small_model(),
    '  ': load_large_model(),
}

#
for name, system in test_systems.items():
    metrics = detector.detect_consciousness(input_data, output_data)

    print(f"\n{name}:")
    print(f"    : {metrics.fractal_dimension:.2f}")
```

```
    print(f"     : {metrics.complexity_coefficient:.2f}")
    print(f"     : {metrics.consciousness_level}/10")
```

5.3

```
0-1:
2-3:      AI
4-6:      AI GPT-2
7-9:     GPT-4
10:        AGI
```

5.1:

```
#     GPT-2
models = {
    'GPT2-small': 'gpt2',
    'GPT2-medium': 'gpt2-medium',
    'GPT2-large': 'gpt2-large',
}

#
```

5.2:

```
#
metrics_before = detector.detect(untrained_model)

#
train(model, data)

#
metrics_after = detector.detect(trained_model)

#
```

---

6 -

-
- GPU
-

6.1

GPU :
  Model → GPU 0 →   →

GPU :
  Model → GPU 0, 1, 2, 3 →   →   →

  :   GPU
EIT-P :

6.2

```
from distributed_training import DistributedEITPTrainer

#
trainer = DistributedEITPTrainer(
    model=model,
    world_size=4,  # 4 GPU
    backend='nccl'  # NVIDIA GPU
)

#
trainer.train(train_data, epochs=10)
```

6.3

```
#  torchrun
torchrun --nproc_per_node=4 production_train.py

# nproc_per_node:    GPU
```

6.1: GPU vs GPU

- GPU - 2 GPU - 4 GPU

6.2: GPU

```
#
watch -n 1 nvidia-smi

#
# • GPU
# •
# •
```

1

: EIT-P

: 1. 1000 2. GPT2-small 3. EIT-P 4. CEP 5.

: - - CEP -

: 2-3

2

: AI “ ”

: 1. 2. consciousness_detection_tool 3. 4. 5.

: - - -

: 1

---

3 CEP

: CEP

: - D 2.7 - - Ω 0

: 1. 2. 3. 4.

: - - -

: 2-4

---

1. **CEP**
   - DOI: 10.5281/zenodo.17301897
   -
2. **EIT-P**
   - DOI: 10.5281/zenodo.17298818
   -

3. **Attention Is All You Need** (Vaswani et al., 2017)
   - Transformer
4. **Scaling Laws for Neural Language Models** (OpenAI, 2020)
   - scaling
5. **Emergent Abilities of Large Language Models** (Google, 2022)
   -

6. : Shannon
7. : Landauer's Principle
8. : Mandelbrot
9. : Edward Lorenz

1-2 : - [ ] CEP - [ ] simple_demo.py - [ ] IEM - [ ] D Ω

2-3 : - [ ] - [ ] - [ ] - [ ]

3-4 : - [ ] - [ ] CEP - [ ] MLOps - [ ]

1. 2. 3. CEP

- GitHub - Markdown -

GitHub Issues

https://github.com/f21211/eitp-real-product/issues

```
[Question] XXX
```

```
1.
2.
3.
4.
```

```
- OS: Linux/Windows/Mac
- Python: 3.x
- PyTorch: x.x
- GPU: /
```

- Email: chen11521@gtiit.edu.cn
- 1-3

- GitHub Discussions
- MD
- examples/

4

Week 1:

```
Mon-Tue:
Wed-Thu: CEP
Fri:
Weekend:
```

Week 2: EIT-P

```
Mon-Tue:
Wed-Thu:
Fri:      demo
Weekend:
```

Week 3:

```
Mon-Tue:
Wed-Thu:
Fri:
Weekend:
```

Week 4:

```
Mon-Tue:
Wed-Thu:
Fri:
Weekend:
```

- ☐ CEP
- ☐
- ☐ CEP

□ EIT-P
□
□
□ CEP

□
□
□
□

---

1. **Bug**: → GitHub Issues
2. : → Pull Request
3. : → Pull Request
4. :

1. **EIT-P** :
2. **CEP** :
3. :
4. : DOI

1. : EIT-P
2. :
3. :

:

: 2025 10 9
:
:
: 2-4