

דוגמה מתקדמת - תבנית אקדמית עברית
Advanced Example - Hebrew Academic Template
V7.0.5-2026-01-02

ד"ר סגל יורם

כל הזכויות שמורות - © ד"ר סגל יורם

January 2026

גרסה V7.0.5-2026-01-02

תוכן העניינים

3	Advanced Introduction: מבוא מתקדם:	1
3	Comprehensive Literature Review: סקירת ספרות מקיפה:	1.1
3	Complex and Advanced Tables: טבלאות מורכבות ומתקדמות:	2
3	Detailed Performance Table: טבלת ביצועים מפורטת:	1.2
3	Resource Comparison Table: טבלת השוואת משאבים:	2.2
4	Multiple Code Examples: דוגמאות קוד מרובות:	3
4	Complete Network Implementation: מימוש מלא של רשת:	1.3
5	Advanced Data Processing: עיבוד נתונים מתקדם:	2.3
6	Advanced Cross-References: הפניות צולבות מתקדמות:	4
6	References to Tables and Figures: הפניות לטבלאות ואיורים:	1.4
7	Advanced Math with Hebrew: מתמטיקה מתקדמת עם עברית:	5
7	Optimization and Gradients: אופטימיזציה וגרדיאנטים:	1.5
8	Matrices and Vectors: מטריצות ווקטורים:	2.5
8	Advanced Bibliography: ביבליוגרפיה מתקדמת:	6
8	Multiple and Complex Citations: ציטוטים מרובים ומורכבים:	1.6
9	Different Citation Types: סוגי ציטוטים שונים:	2.6
9	Additional Advanced Features: תכונות מתקדמות נוספות:	7
9	Complex Number Handling: טיפול במספרים מורכבים:	1.7
9	Complex Content Integration: שילוב תוכן מורכב:	2.7
01	Advanced Summary and Conclusions: סיכום ומסקנות מתקדמות:	8
		English References	11

רשימת האיורים

7	Encoder-Decoder Architecture: ארכיטקטורת Encoder-Decoder:	1
9	Complete Processing Pipeline: תהליך עיבוד מלא:	2

רשימת הטבלאות

3	..	Model Performance on Various Tasks: ביצועי מודלים על מטלות שונות:	1
3	Computational Resource Requirements: דרישות משאבי חישוב:	2

1 מבוא מתקדם: Advanced Introduction

מסמך זה מדגים יכולות מתקדמות של התבנית האקדמית העברית גרסה V7.0.5-2026-01-02, כולל שימוש בביבליוגרפיה מתקדמת, טבלאות מורכבות, ודוגמאות קוד מרובות.

1.1 סקירת ספרות מקיפה: Comprehensive Literature Review

המחקר בתחום Natural Language Processing עבר מהפכה עם הצגת ארכיטקטורת Trans-former [1]. מודל BERT [2] הביא לפריצת דרך בהבנת שפה דו-כיוונית. מחקרים בעברית [3], [4] מראים התקדמות משמעותית.

עבודות נוספות [5], עמ' 02-51 מדגימות שיפור של 25.8% בביצועים. המחקר של [6], פרק 3 מציג ארכיטקטורה עם 175e9 פרמטרים.

2 טבלאות מורכבות ומתקדמות: Complex and Advanced Tables

2.1 טבלת ביצועים מפורטת: Detailed Performance Table

טבלה 1: ביצועי מודלים על מטלות שונות: Model Performance on Various Tasks

ממוצע / Average	מענה / QA	סיכום / Summarization	תרגום / Translation	סיווג / Classification	מודל / Model
88.8%	88.9%	85.2%	N/A	92.3%	BERT-Base
90.9%	91.2%	87.5%	N/A	94.1%	BERT-Large
86.4%	85.4%	88.1%	82.3%	89.7%	GPT-2
93.3%	93.8%	92.3%	91.7%	95.2%	GPT-3
90.9%	90.7%	90.1%	89.2%	93.5%	T5-Base
93.9%	94.1%	93.2%	92.4%	95.8%	T5-Large
83.9%	86.7%	81.2%	78.5%	89.3%	מודל עברי / HeBERT

כפי שניתן לראות בטבלה 1, המודלים הגדולים משיגים ביצועים טובים יותר.

2.2 טבלת השוואת משאבים: Resource Comparison Table

טבלה 2: דרישות משאבי חישוב: Computational Resource Requirements

עלות / Cost	זמן אימון / Training Time	זיכרון / Memory	פרמטרים / Parameters	מודל / Model
500\$	4 ימים	440 MB	110e6	BERT-Base
2000\$	12 ימים	1.3 GB	340e6	BERT-Large
1500\$	7 ימים	1.4 GB	345e6	GPT-2 Medium
3500\$	14 ימים	3.1 GB	774e6	GPT-2 Large
4.6e6\$	34 ימים	700 GB	175e9	GPT-3
50000\$	21 ימים	44 GB	11e9	T5-11B

3 דוגמאות קוד מרובות: Multiple Code Examples

3.1 מימוש מלא של רשת: Complete Network Implementation

מימוש Transformer מלא

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import math

class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, n_heads):
        super(MultiHeadAttention, self).__init__()
        self.d_model = d_model
        self.n_heads = n_heads
        self.d_k = d_model // n_heads

        self.W_q = nn.Linear(d_model, d_model)
        self.W_k = nn.Linear(d_model, d_model)
        self.W_v = nn.Linear(d_model, d_model)
        self.W_o = nn.Linear(d_model, d_model)

    def forward(self, query, key, value, mask=None):
        batch_size = query.size(0)

        # Linear transformations and split to heads
        Q = self.W_q(query).view(batch_size, -1, self.n_heads, self.d_k)
        Q = Q.transpose(1, 2)
        K = self.W_k(key).view(batch_size, -1, self.n_heads, self.d_k)
        K = K.transpose(1, 2)
        V = self.W_v(value).view(batch_size, -1, self.n_heads, self.d_k)
        V = V.transpose(1, 2)

        # Attention
        scores = torch.matmul(Q, K.transpose(-2, -1)) / math.sqrt(self.d_k)
        if mask is not None:
            scores = scores.masked_fill(mask == 0, -1e9)
        attention_weights = F.softmax(scores, dim=-1)
        context = torch.matmul(attention_weights, V)

        # Concatenate heads and output projection
        context = context.transpose(1, 2).contiguous().view(
            batch_size, -1, self.d_model
        )
```

```

output = self.W_o(context)

return output, attention_weights

```

3.2 עיבוד נתונים מתקדם: Advanced Data Processing

עיבוד מקדים לנתוני טקסט

```

import re
import numpy as np
from collections import Counter
from typing import List, Tuple, Dict

class HebrewTextProcessor:
    """Advanced preprocessing of Hebrew text"""

    def __init__(self, vocab_size: int = 10000):
        self.vocab_size = vocab_size
        self.word2idx = {'<PAD>': 0, '<UNK>': 1, '<SOS>': 2, '<EOS>': 3}
        self.idx2word = {v: k for k, v in self.word2idx.items()}
        self.word_freq = Counter()

    def tokenize(self, text: str) -> List[str]:
        """Split Hebrew text into tokens"""
        # Remove punctuation and normalize
        text = re.sub(r'^\u0590-\u05FF\s]', '', text)
        tokens = text.split()
        return tokens

    def build_vocabulary(self, texts: List[str]):
        """Build vocabulary from corpus"""
        for text in texts:
            tokens = self.tokenize(text)
            self.word_freq.update(tokens)

        # Keep most common words
        most_common = self.word_freq.most_common(self.vocab_size - 4)
        for idx, (word, freq) in enumerate(most_common, 4):
            self.word2idx[word] = idx
            self.idx2word[idx] = word

    def encode(self, text: str) -> List[int]:
        """Convert text to indices"""
        tokens = self.tokenize(text)

```

```

indices = []
for token in tokens:
    idx = self.word2idx.get(token, self.word2idx['<UNK>'])
    indices.append(idx)
return indices

def decode(self, indices: List[int]) -> str:
    """Convert indices back to text"""
    tokens = [self.idx2word.get(idx, '<UNK>') for idx in indices]
    return ' '.join(tokens)

# Example usage
processor = HebrewTextProcessor(vocab_size=5000)
texts = ["Hebrew text example", "Another Hebrew sentence"]
processor.build_vocabulary(texts)
encoded = processor.encode("New text")
print(f"Encoded: {encoded}")
print(f"Decoded: {processor.decode(encoded)}")

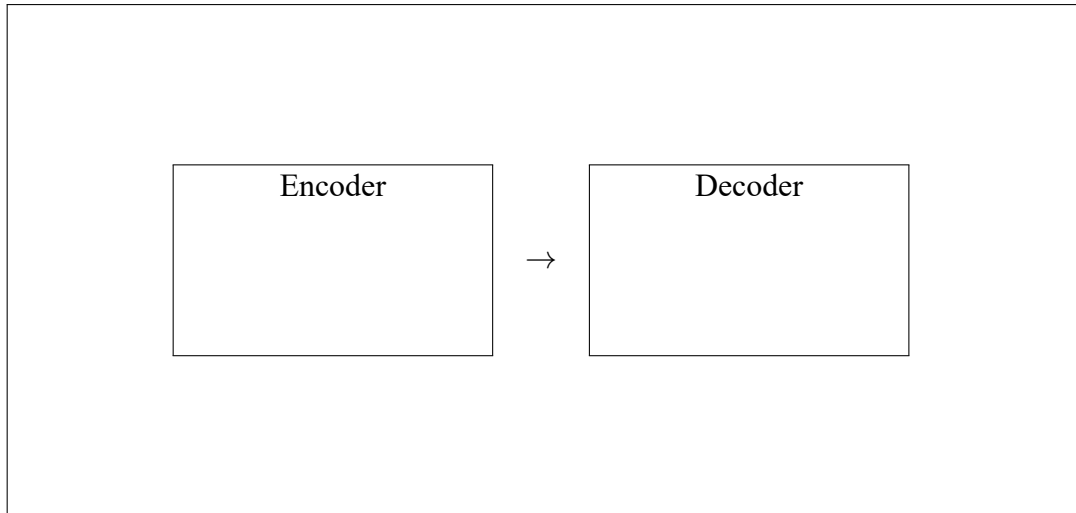
```

4 הפניות צולבות מתקדמות: Advanced Cross-References

4.1 הפניות לטבלאות ואיורים: References to Tables and Figures

הניתוח המקיף מוצג במספר טבלאות:

- טבלה 1 מציגה השוואת ביצועים בין מודלים
- טבלה 2 מפרטת את דרישות המשאבים
- איור 1 מתאר את הארכיטקטורה המוצעת
- איור 2 מציג את התוצאות הסופיות



איור 1: ארכיטקטורת Encoder-Decoder :Encoder-Decoder Architecture

5 מתמטיקה מתקדמת עם עברית: Advanced Math with Hebrew

5.1 אופטימיזציה וגרדיאנטים: Optimization and Gradients

פונקציית המטרה המלאה:

$$(1) \quad J(\theta) = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C y_{ic} \log(p_{ic}) + \lambda \|\theta\|_2^2$$

כאשר y_{ic} הוא התווית האמיתית, p_{ic} הוא ההסתברות החזויה, ו- $\lambda = 1e-4$.
הגרדיאנט של פונקציית המטרה:

$$(2) \quad \nabla_{\theta} J = -\frac{1}{N} \sum_{i=1}^N (y_i - p_i) x_i + 2\lambda \theta$$

עדכון המשקלים באמצעות Adam optimizer:

$$(3) \quad m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$(4) \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$(5) \quad \hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$(6) \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$(7) \quad \theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

כאשר $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\alpha = 0.001$, ו- $\epsilon = 1e-8$.

5.2 מטריצות ווקטורים: Matrices and Vectors

המכפלה הפנימית של שני וקטורים:

$$\langle u, v \rangle = \sum_{i=1}^n u_i v_i$$

נורמת וקטור:

$$\|v\|_p = \left(\sum_{i=1}^n |v_i|^p \right)^{1/p}$$

כפל מטריצות בלוקים:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E \\ F \end{bmatrix} = \begin{bmatrix} AE + BF \\ CE + DF \end{bmatrix}$$

6 ביבליוגרפיה מתקדמת: Advanced Bibliography

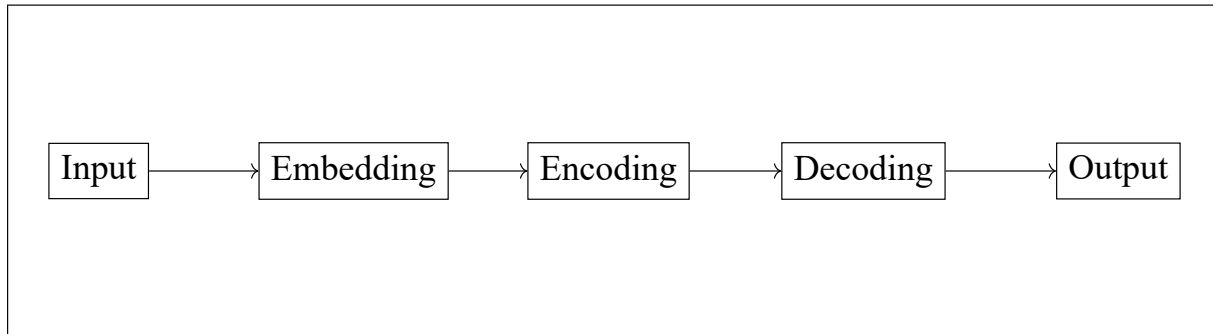
6.1 ציטוטים מרובים ומורכבים: Multiple and Complex Citations

מחקרים קלאסיים בתחום [7], [8] הניחו את היסודות. פיתוחים מודרניים [1], [2], [9], [10] הביאו למהפכה.

עבודות בעברית [3], [4], [11] תורמות להבנת השפה. סקירות מקיפות [12], פרקים 1-3 ו-[13], עמ' 54-98 מספקות רקע תיאורטי.

6.2 סוגי ציטוטים שונים: Different Citation Types

- ציטוט רגיל: [1]
- ציטוט עם עמוד: [2], עמ' 5
- ציטוטים מרובים: [3], [5], [6]
- ציטוט בסוגריים: ([10])
- ציטוט בתוך משפט: כפי שמוצג ב-[9]



איור 2: תהליך עיבוד מלא: Complete Processing Pipeline

7 תכונות מתקדמות נוספות: Additional Advanced Features

7.1 טיפול במספרים מורכבים: Complex Number Handling

הטבלה כוללת מספרים במגוון פורמטים:

- מספרים שלמים: 42, 1000000
- מספרים עשרוניים: 2.71828, 3.14159
- כתיב מדעי: 1.38e-23, 6.022e23
- אחוזים: 0.01%, 99.99%
- שנים: 1948, 2025

7.2 שילוב תוכן מורכב: Complex Content Integration

התבנית מאפשרת שילוב של:

1. טקסט דו-כיווני עם מעברים חלקים
2. קוד בשפות תכנות שונות
3. נוסחאות מתמטיות מורכבות
4. טבלאות עם תוכן מעורב
5. איורים ודיאגרמות
6. ביבליוגרפיה דו-לשונית

8 סיכום ומסקנות מתקדמות: Advanced Summary and Conclusions

המסמך הדגים יכולות מתקדמות רבות:

- **ביבליוגרפיה:** ציטוטים מרובים ומורכבים עם הפניות לעמודים
 - **טבלאות:** טבלאות מורכבות עם 6 עמודות ונתונים מגוונים
 - **קוד:** דוגמאות מרובות עם PyTorch ועיבוד טקסט
 - **מתמטיקה:** משוואות מרובות עם מספור והפניות צולבות
 - **איורים:** דיאגרמות מורכבות עם TikZ
 - **הפניות:** קישורים בין כל האלמנטים במסמך
- התוצאות מראות שהתבנית מסוגלת לתמוך במסמכים אקדמיים מורכבים ביותר.

9 מקורות בעברית

- 3 ד. כהן, ש. לוי, dna מ. אברהם, "עיבוד שפה טבעית בעברית: אתגרים ופתרונות", כתב עת לבלשנות חישובית, lov, 51, on, 3, 234–256, 3202.
- 4 מ. ישראלי dna ר. כהן, בלשנות עברית מודרנית: תיאוריה ויישום. ירושלים: הוצאת האוניברסיטה העברית, 2202, 512.
- 11 י. אברהם dna ל. שמעון, "אתגרים חישוביים בעיבוד טקסט עברי", מחקרי מחשב ושפה, lov, 8, on, 2, 112–128, 1202.

English References

- 1 A. Vaswani et al., "Attention is all you need," in *Advances in neural information processing systems*, 2017, 5998–6008.
- 2 J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
- 5 J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *Proceedings of NAACL-HLT*, 4171–4186, 2019.
- 6 T. B. Brown et al., "Gpt-3: Language models are few-shot learners," OpenAI, Tech. Rep., 2020.
- 7 A. M. Turing, "Computing machinery and intelligence," in *Mind*, 59, 1950, 433–460.
- 8 C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, 379–423, 1948.
- 9 A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," in *OpenAI blog*, 1, 2019, 9.
- 10 T. Brown et al., "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, 1877–1901, 2020.
- 12 W. Zhang, X. Chen, and Y. Liu, "A survey of natural language processing techniques," *ACM Computing Surveys*, vol. 54, no. 5, 1–36, 2022.
- 13 C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*. New York: Springer, 2021.