

# **דוגמה מתקדמת - תבנית אקדמית עברית**

## **Advanced Example - Hebrew Academic Template v5.0**

ד"ר סgal יoram

כל הזכויות שמורות - © Dr. Segal Yoram

November 2025

גרסה 0.5 - מתקדמת

## תוכן העניינים

3	מבוא מתקדם: Advanced Introduction	1
3	סקירה ספרות מקיפה: Comprehensive Literature Review	1.1
3	טבלאות מורכבות וمتקדמיות: Complex and Advanced Tables	2
3	טבלת ביצועים מפורטת: Detailed Performance Table	2.1
3	טבלת השוואת משאבים: Resource Comparison Table	2.2
4	דוגמאות קוד מרובות: Multiple Code Examples	3
4	IMPLEMENTATION: Complete Network Implementation	3.1
5	עיבוד נתונים מתקדם: Advanced Data Processing	3.2
6	הפניות צולבות מתקדמות: Advanced Cross-References	4
6	הפניות לATABLES AND FIGURES: References to Tables and Figures	4.1
6	מתמטיקה מתקדמה עם עברית: Advanced Math with Hebrew	5
6	אופטימיזציה וגרדיינטים: Optimization and Gradients	5.1
7	מטריצות ווקטורים: Matrices and Vectors	5.2
7	ביבליוגרפיה מתקדמת: Advanced Bibliography	6
7	ציטוטים מרובים ומורכבים: Multiple and Complex Citations	6.1
7	סוגי ציטוטים שונים: Different Citation Types	6.2
8	תכונות מתקדמות נוספות: Additional Advanced Features	7
8	טיפול במספרים מורכבים: Complex Number Handling	7.1
8	שילוב תוכן מורכב: Complex Content Integration	7.2
9	סיכום ומסקנות מתקדמות: Advanced Summary and Conclusions	8
10	מקורות בעברית . . . . .	9
10	English References . . . . .	10

## רשימת האיורים

6	ארQUITECTURE Encoder-Decoder :Encoder-Decoder	1
8	תהליך עיבוד מלא: Complete Processing Pipeline	2

## רשימת הטבלאות

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

# 1 מבוא מתקדם: Advanced Introduction

מסמך זה מדגים יכולות מתקדמות של התבנית האקדמית העברית גרסה 0.5, כולל שימוש בביבליוגרפיה מתקדמת, טבלאות מורכבות, ודוגמאות קוד רבות.

## 1.1 סקירת ספרות מקיפה: Comprehensive Literature Review

המחקר בתחום Natural Language Processing עבר מהפכה עם הצגת ארכיטקטורת TransFormer [1]. מודל BERT [2] הוביל לפירצת דרך בהבנת שפה דו-cybernetic. מחקרים בעברית [3], [4] מראים התקדמות משמעותית. בעבודות נוספות [5], עמ' 02-51 מוגדרות שיפור של 25.8% ביצועים. המאמר של 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

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

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

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

### 3 דוגמאות קוד מרובות: Multiple Code Examples

#### 3.1 מימוש מלא של רשת: Complete Network Implementation

##### מימוש מלא Transformer

```
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)

        # ישארה קולחנות ראייתנו צמרופסנרט #
        Q = self.W_q(query).view(batch_size, -1, self.n_heads, self.d_k) \
            .transpose(1, 2)
        K = self.W_k(key).view(batch_size, -1, self.n_heads, self.d_k) \
            .transpose(1, 2)
        V = self.W_v(value).view(batch_size, -1, self.n_heads, self.d_k) \
            .transpose(1, 2)

        # קפ
        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 לנוטוני טקסט

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

class HebrewTextProcessor:
    """ירבעעטסזטלשבםדקתטפ-דוביעמדו"""

    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]:
        """מינומיסאלפירבעטסזטלואיף"""
        # לוחרנווקוסיפינומיסחרסה
        text = re.sub(r'[^\u0590-\u05FF\s]', '', text)
        tokens = text.split()
        return tokens

    def build_vocabulary(self, texts: List[str]):
        """סופרוקטטסילימראצואתוי ינב"""
        for text in texts:
            tokens = self.tokenize(text)
            self.word_freq.update(tokens)

        # רתויבתוזונהם לימהתריחס
        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]:
        """מיצדנייאלבטסזטטמרהה"""
        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:
        """טסזטטלברוזטטסזטטניאטמרה"""
        tokens = [self.idx2word.get(idx, '<UNK>') for idx in indices]
        return ''.join(tokens)
```

---

כל הזכויות שמורות © Dr. Segal Yoram

## 4 הפניות צולבות מתקדמות: Advanced Cross-References

### 4.1 הפניות לdatableות ואיורים: References to Tables and Figures

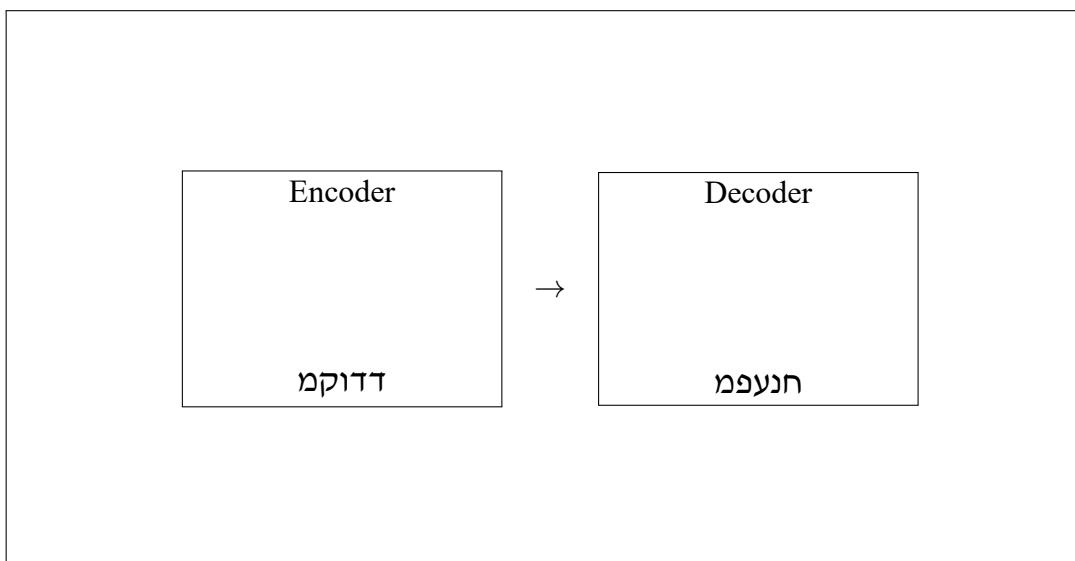
הנition המקי' מוצג במספר טבלאות:

- 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 וגרדיינטים: 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 התוויות האמיתית,  $p_{ic}$  הוא ההסתברות החזויה, ו- $\lambda = 1e - 4 - 4$  הגרדיינט 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

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

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

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

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

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

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

## 5.2 מטריצות וקטורים:

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 בлокים:

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

## 6 ביבליוגרפיה מתקדמת:

### 6.1 ציטוטים מרובים ומורכבים:

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

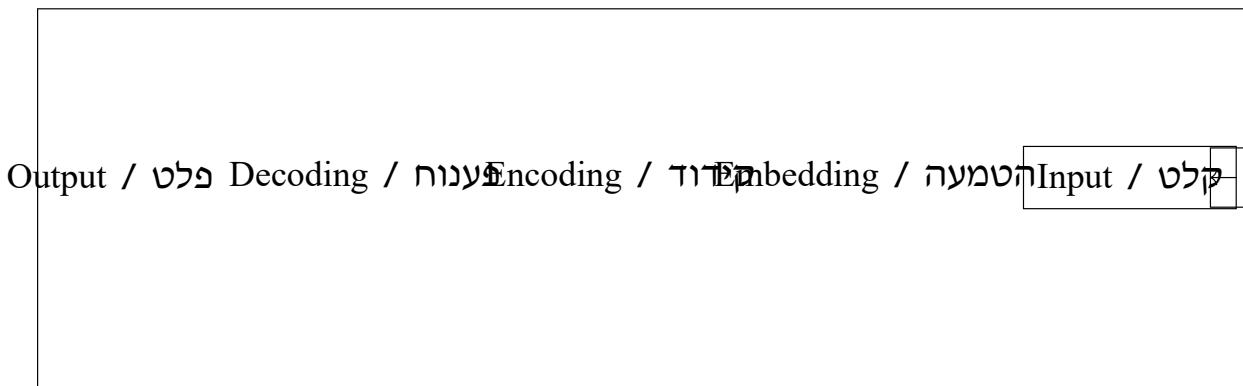
עבודות בעברית [3], [4], [11] תורמות להבנת השפה. סקירות מקיפות [12], פרקים 1-3, 1-[13], עמ' 54-98 מספקות רקע תיאורטי.

### 6.2 סוגים של ציטוטים:

- ציטוט רגיל: [1]

- ציטוט עם עמוד: [2], עמ' 5

- ציטוטים מרובים: [3], [5], [6]
- ציטוט בסוגריים: ([10])
- ציטוט בתוך משפט: כפי שמצוג ב-[9]



איור 2: תהליך עיבוד מלא: Complete Processing Pipeline

## 7 **תכונות מתקדמות נוספות:** Additional Features

### 7.1 **טיפול במספרים מורכבים:** Complex Number Handling

הטבלה כוללת מספרים במגוון פורמטים:

- מספרים שלמים: 42, 1000000

- מספרים עשרוניים: 3.14159, 2.71828

- כתיב מדעי: 6.022e23, 1.38e-23

- אחוזים: 99.99%, 0.01%

- שנים: 2025, 1948

### 7.2 **שילוב תוכן מורכב:** Complex Content Integration

התבנית מאפשרת שילוב של:

1. טקסט דו-כיווני עם מעברים חלקיים

2. קוד בשפות תכנות שונות

3. נוסחאות מתמטיות מורכבות

4. טבלאות עם תוכן מעורב

5. איורים ודיגרמות

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," כתוב עת לכלי לשונות חישוביות, 3, 234–256 .on ,51 .lov .3202
- 4 מ. ישראלי dna ר. כהן, *כלי לשונות עברית מודרנית: תיאוריה ויישום*. ירושלים: הוצאת האוניברסיטה העברית, 2202, 512 .
- 11 ג. אברהם dna ל. שמעון, "אתגרים חישוביים בעיבוד טקסט עברי," מחקרים מחשב ושפה, 2, 112–128 .on ,8 .lov .1202

## 10 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.