ChatGPT

Deep Past Challenge – Akkadca'dan İngilizce'ye Makine Çevirisi Yarışması İçin Derin Analiz ve Kazanma Odaklı Rehber

Görev tanımı ve değerlendirme metriği

Bu yarışmanın temel amacı, **Latin harfleriyle çevriyazımı (transliteration) verilmiş Eski Asurca/ Akkadca metinleri İngilizceye çevirmek** üzere bir makine çevirisi sistemi geliştirmektir. Yarışma anlatımı, "4.000 yıllık ticari/kayıt metinleri" bağlamını vurgular; cuneiform tabletler üzerine yazılmış iş kayıtlarının (kontrat, mektup, borç/ödeme vb.) modern İngilizceye çevrilmesi hedeflenir. [1]

Değerlendirme, tek bir metrik yerine iki klasik otomatik çeviri metriğinin **geometrik ortalaması** ile yapılır. Resmî tanıma göre skor:

$$\text{Score} = \sqrt{\text{BLEU} \times \text{chrF++}}$$

Ayrıca yarışma değerlendirmesi, **korpus düzeyinde yeterli istatistikleri (sufficient statistics) satırlar üzerinde biriktirip** BLEU ve chrF++'ı bu birikimli sayımlardan hesaplar (yani "cümle başına skorların ortalaması" gibi değil, klasik corpus-level yaklaşım). [2]

BLEU (Bilingual Evaluation Understudy): Papineni vd. (2002) tarafından önerilen; aday çeviri ile referans çeviri arasındaki **kelime n-gram örtüşmelerinin (modified precision)** geometrik ortalamasını ve **brevity penalty (BP)** terimini birleştiren metriktir. Tipik form (4-gram, eşit ağırlıklar) şu şekildedir:

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right), \quad \sum w_n = 1$$

Burada p_n "modified n-gram precision", BP ise adayın aşırı kısa olmasını cezalandıran kısalık cezasıdır. [3]

chrF / chrF++: Popović (2015) tarafından önerilen **karakter n-gram F-score** ailesidir. chrF; karakter n-gram precision/recall üzerinden β ile (çoğunlukla $\beta = 2$) recall'a daha fazla ağırlık vererek F-skoru üretir. Özet form:

$$chrF = (1 + \beta^2)\frac{P \cdot R}{\beta^2 \cdot P + R}$$

Popović'nin devam niteliğindeki "++" varyantı (chrF++), karakter n-gramlara ek olarak **kelime n-gramlarını** (pratikte kelime unigram/bigram) da hesaba katar; bu yarışma özelinde kullanılan "chrF++" ifadesi de bunu işaret eder. [4]

Bu metrik seçimi, özellikle bu yarışmada şu sonucu doğurur: **anlamsal olarak doğru ama farklı sözcüklerle paraphrase edilmiş çeviriler** BLEU/chrF++'ta ciddi puan kaybedebilir; bu durum hem katılımcı tartışmalarında hem de metrik açıklamalarında açıkça vurgulanmıştır. [5]

Veri seti anatomisi, riskler ve güçlü ön işleme yaklaşımı

Yarışma verisi, **8.000+ Eski Asurca cuneiform metnin çevriyazımını** ve "kapsamlı metadata"yı içerdiğini belirtir; bunların bir altkümesinde İngilizce çeviri hizalamaları bulunur. [6]

Dosyalar ve şema (pratikte Kaggle dizin yapısı)

Kaggle Notebook'larda sık kullanılan yol:
`/kaggle/input/deep-past-initiative-machine-translation/` [7]

Bu paket içinde kamuya açık örnek/yardımcı dosyalar şu şekilde raporlanıyor:

- **train.csv**: 1.561 satır; temel eğitim paralel veri. Sütunların `oare_id`, `transliteration`, `translation` olduğu ve tipik EDA'larda eksik değer görülmediği raporlanmış durumda. [8]
- **test.csv**: "küçük örnek test" dosyasıdır; skorlanırken bunun yerine **tam gizli test seti** kullanılır. Ayrıca pratikte örnek testin 4 satır olduğu sıkça görülür. [9]
- **sample_submission.csv**: doğru formatta örnek teslim dosyasıdır (genellikle 4 satır örnek). [10]
- **published_texts.csv**: yaklaşık **7.953** çevriyazım örneği içerdiği, ancak çeviri (ground truth) içermediği belirtilir; bazı tartışmalarda "transliteration_orig" ve "transliteration" gibi iki varyant sütunundan bahsedilir. [11]
- **Sentences_Oare_FirstWord_LinNum.csv**: "cümle hizalama" için yardımcı format; ayrıca tartışmalarda **9.782 alignment** rakamı geçer ve eğitim/değerlendirme için "en faydalı format" olduğu söylenir. [12]
- **OA_Lexicon_eBL.csv**: çevriyazım kelimelerini ve "lexical equivalent" bilgisini içeren bir **Eski Asurca sözlük/lexicon**. [13]
- **resources.csv**: ek veri/bağlantı kaynakları listesi olarak tanımlanır. [14]

Not: Bazı Notebook loglarında **bibliography.csv / publications.csv** gibi dosyalar da geçtiği görülüyor; bu, yarışma paketinde ek bibliyografik veya yayın meta verisi olabileceğine işaret eder. [15]

Veri kalitesi ve "bu yarışmayı zorlaştıran" tipik problemler

Veri, akademik yayın gelenekleri nedeniyle **birden fazla format/konvansiyonun karışımı** olarak sunuluyor; bu durum, özellikle çevriyazımda (diakritikler, determinatifler, boşluk gösterimi) tutarsızlık üretebiliyor. Resmî kanallarda, "bir yüzyıla yayılan ve birçok dile dağılan" çalışmaların farklı çevriyazım gelenekleri oluşturduğu; bunun yarışmada ilk büyük darboğaz olduğu vurgulanır. [16]

Öne çıkan risk kümeleri:

- **Named entity (özel isim) davranışı**: İsimler/yer adları farklı yazım varyantlarıyla gelebilir; bu hem MT kalitesini düşürür hem de BLEU/chrF++ gibi yüzey metriklerini sert etkiler. Bu yüzden ek onomasticon (adlar sözlüğü) gibi kaynakların ekleneceği de duyurulmuştur. [17]
- **Gap/hasar gösterimi**: Çevriyazımda hasarlı kısımlar farklı biçimlerde (ör. `xx`, `...`, farklı braketler) gelebilir; referans çeviride ise `<gap>` / `<big_gap>` gibi özel token'lar bulunabilir ve bunların doğru üretilmesi skor için kritiktir. Bu konu, yüksek skor arayanların özellikle tartıştığı "gotcha"lardan biridir. [18]
- **Train–published_texts biçim farklılıkları**: Bazı tartışmalarda train'de `()` ile görülen yapıların published_texts'te `{}` ile görünebildiğinden bahsedilir; bu, test-time normalizasyonu zorunlu kılar. [19]

- **Hizalama/misalignment riski**: Sentences_*.csv ile train.csv arasında problemli hizalamalar olduğu ve belirli sayıda metinde hata görüldüğü tartışılmıştır; yanlış hizalanmış cümle çiftleri eklemek LB'yi düşürebilir. [20]

Metrik-optimal ön işleme stratejisi

Bu yarışmada "daha iyi çeviri" ile "daha yüksek BLEU/chrF++" her zaman aynı şey değildir; metrik **string benzerliğine** aşırı duyarlı olduğundan, ön işleme hedefiniz çoğu zaman "anlamı en iyi çözmek" değil "referans üslubunu taklit etmek" olur. [5]

Bu nedenle önerilen pratik pipeline:

1) **Unicode ve whitespace standardizasyonu**: NFC normalizasyonu, NBSP/zero-width gibi görünmez karakterlerin kaldırılması, çoklu boşlukların tekleştirilmesi (özellikle diakritikli harflerde aynı karakterin farklı Unicode bileşimleri olabiliyor). Bu tür standardizasyon, hem tokenizasyon tutarlılığı hem de sacreBLEU gibi araçlarla metrik raporlamada karşılaştırılabilirlik için kritiktir. [21]

2) **Gap normalizasyonu**: Source transliteration içindeki `xx`, `x x`, `[...]`, `...` gibi gösterimleri bir kanonik etiket kümesine indirgeme (örn. `<gap>`, `<big_gap>`). Bu, modelin hem eğitimde hem testte aynı "hasar dili"ni görmesini sağlar. [22]

3) **Determinatif/işaretleme standardı**: `{}` / `()` / köşeli parantez gibi yapıların yarışma referanslarına en yakın forma dönüştürülmesi. Bu adım özellikle "hidden test formatı train'den farklı olabilir mi?" tartışmalarında kritik görülüyor. [23]

4) **Özel isim kontrol katmanı**: OA_Lexicon ve (varsa) onomasticon gibi kaynaklarla, özel isimlerin varyantlarını "kopyalama/koruma" yaklaşımıyla ele almak. (Modelin özel isimleri yanlış "İngilizceleştirmesi" veya başka isimle değiştirmesi hem BLEU'yu hem chrF++'ı zedeler.) [24]

Tokenizasyon: ByT5 mi, SentencePiece/BPE mi?

Kaynak dil "Latin çevriyazımı" olduğu için klasik BPE/SentencePiece hâlâ uygulanabilir; ancak yarışmanın pratik gerçekliği (diakritikler, bozuk OCR benzeri gürültü, tutarsız ayrımlar) nedeniyle **byte/char-seviyesinde** modelleme güçlü bir avantaj sağlar. ByT5, UTF-8 byte dizileri üzerinde çalışarak **tokenizer bağımlılığını kaldırır** ve gürültüye daha dayanıklı olduğunu raporlar. Bu, Eski Asurca çevriyazımı gibi "noisy text" durumlarında doğrudan hedefe uyar. [25]

Buna karşılık, subword yaklaşımı kurmak isterseniz:

- **SentencePiece** (Unigram veya BPE) ham metinden öğrenebilir; dil-bağımsız tokenizer/detokenizer sunar. [26]
- BPE'nin NMT'de OOV sorunlarını azalttığı klasik gösterim Sennrich vd. tarafından anlatılır. [27]

Bu yarışmaya özgü karar: **üst sıralardaki kamu çözümlerinin büyük kısmı ByT5 varyantlarını kullanıyor**; bu da (en azından public LB) için token-free yaklaşımın pratikte avantajlı olduğuna dair güçlü bir işaret. [28]

Yarışma ortamı, kurallar, teslim formatı ve pratik kısıtlar

Bu yarışma, tipik bir "code competition" düzeninde, **internet kapalı** koşuluyla çalışmayı zorunlu kılar; yani Notebook içinde dışarıdan paket/model indirmek mümkün değildir. Bu yüzden kullanılacak ek veriler ve pre-trained modeller, Kaggle'a "dataset" olarak eklenmiş olmalıdır. [29]

Yarışma sayfasında, **serbest ve kamusal** (freely & publicly available) dış verinin ve **pre-trained modellerin** kullanılmasına izin verildiği belirtilir; fakat bunların kaynak/erişim açısından tüm yarışmacılara açık olması beklenir. [30]

Çalışma süresi açısından, CPU/GPU Notebook için **9 saat** sınırı olduğu belirtilir; bu da "tek notebook içinde devasa eğitim" yerine, pratikte (a) önceden eğitilmiş checkpoint'ler veya (b) kısa fine-tune + güçlü inference odaklı yaklaşımları teşvik eder. [31]

Teslim formatı:

- Çıktı dosyası `submission.csv` olmalı ve yarışma formatına uymalıdır. [32]
- `test.csv` yalnızca bir **örnek**; skorlamada bunun yerine gerçek test seti yerleştirilir. Bu, hatalı bir stratejinin "4 satırlık testte çalışıyor" görünmesine rağmen gerçek testte çökmesine yol açabilir. [33]

Lider tablo (leaderboard) dinamiği:

- Public LB'nin testin yaklaşık **%34'ü** ile hesaplandığı; final (private) skorun kalan kısım üzerinden belirleneceği açıkça ifade edilmiştir. Bu nedenle "public'e optimize hack"lerin private'ta düşmesi beklenir. [34]

Bu yarışma, Google [35] çatısı altındaki platformun tipik kurallarını izler ve organizasyon/sponsorluk tarafında XTX Markets [36] desteği gibi unsurların yer aldığı belirtilmiştir. [37]

Tartışma ve lider tablo sinyalleri

Yarışma hâlen aktif (Şubat 2026 itibarıyla kapanmamış) olduğundan "kazanan" (final private LB) çözümler henüz yoktur; bu bölüm, **public LB'de görülen güçlü yaklaşımlar** ile **tartışmalarda tekrar eden teknik dersleri** birleştirir. [38]

Tartışmalardan çıkan yüksek kaldıraçlı dersler

Metrik gerçeği: "anlam" değil "eşleşme" ödüllendiriliyor. Katılımcılar açıkça BLEU/chrF++'ın semantic doğruluktan çok yüzey n-gram benzerliğini ödüllendirdiğini; LLM tabanlı paraphrase/post-edit'in bile skoru düşürebildiğini belirtiyor. Bu, eğitimde ve inference'ta "daha deterministik, daha şablonvari" çıktılara yönelmeyi teşvik eder. [5]

Gap token uyumu bir "gotcha". Bazı kullanıcılar, tahminlerde `<gap>` / `<big_gap>` etiketlerinin doğru taşınmaması veya sayıca uyumsuz olması nedeniyle skorun zarar gördüğünü; model eğitiminde bu işaretleri canonical hale getirmenin önemli olduğunu yazıyor. [39]

Sentence-level hizalama potansiyel ama riskli. Sentences_*.csv cümle-segment hizalaması için "en faydalı format" olarak anılsa da, yanlış hizalanmış örnekler eklemek public LB'yi düşürebiliyor. Ayrıca

train.csv ile bu yardımcı dosya arasında sorunlu örnekler bulunduğu da tartışılıyor. Bu, “daha çok veri” ile “daha iyi veri” ayrımını kritik hale getirir. [40]

RL / preference-based yöntemler pratikte zayıf sinyal veriyor olabilir. Resmî kanallarda, supervised fine-tuning’in stabil loss verdiği; RL ya da tercih temelli yaklaşımların “output non-conformance” yüzünden ödül sinyali üretemeyebildiği belirtiliyor. Bu yarışmada, önce “format konformansı”nı çözmeden RL’nin verimsiz kalması sürpriz değil. [41]

Public LB’de görülen “yüksek skor” çizgisi ve örnek notebook’lar

Topluluk özetlerinde public model tavanının ~35 civarı olduğu ve bunun üstüne çıkmanın zorlaştığı ifade ediliyor; public LB’de 35.1 civarı skorlar raporlayan notebook’lar mevcut. [42]

Aşağıdaki tablo, Şubat 2026 itibarıyla arama sonuçlarında görünen ve public skor bildiren bazı güçlü notebook’ları (tam liste değildir) yaklaşımlarıyla birlikte özetler:

Notebook / Çalışma	Ana fikir	Raporlanan public skor	Kanıt
ByT5 + Optuna Tuning + Chunked Beam Search	ByT5 + hiperparametre araması + (uzun metinlerde) “chunked” beam search	35.1	[43]
Deep Past Challenge ver 3	ByT5 tabanlı inference pipeline	35.0	[44]
BYT-Ensemble | Script	Birden çok ByT5 checkpoint’i ile ensemble	34.5	[45]
Akkadian M2M Inference	M2M tabanlı inference (karşılaştırma için iyi)	27.5	[46]
Hybrid best Akkadian	Farklı “hybrid” denemeler	22.0 civarı (notebook sürümüne bağlı)	[47]

Bu tabloda görülen örüntü (kamu LB bağlamında) oldukça nettir: **ByT5 tabanlı ve özellikle ensemble/ tuning içeren çözümler**, modern çokdilli token düzeyli modellerin basit inference’ına kıyasla daha yüksek skorlar raporlar. Bu bir “private garanti” değildir; ancak public sinyal açısından güçlüdür. [48]

Model stratejisi ve sistem tasarımı

Bu bölüm, “baseline → top-6 hedefli” bir yol haritasını, model ailesi seçimlerini ve yarışmaya özgü ayarları birlikte verir.

Baseline model ailesi seçimi

ByT5 (byte-level seq2seq): Bu yarışma özelinde en rasyonel başlangıç noktasıdır; çünkü ByT5, text’i UTF-8 byte seviyesinde işler, tokenizer bağımlılığını kaldırır ve gürültü/yanlış yazım gibi bozulmalara daha dayanıklı olabileceğini gösterir. [25]

MarianMT: Hugging Face [49] dokümantasyonu, MarianMT’nin Transformer encoder-decoder olduğunu ve MT için tasarlandığını açıklar; ancak bu yarışmada kaynak dil “modern doğal dil” olmadığı için

(çevriyazım/gap/determinatif) doğrudan uyum daha zayıf olabilir. MarianMT, yine de "hızlı karşılaştırma baseline" için değerlidir. [50]

mBART / multilingual denoising pretraining: mBART, düşük kaynaklı MT'de büyük BLEU kazanımları raporlayan bir multilingual seq2seq ön-eğitim hattıdır; fakat Eski Asurca çevriyazımı, mBART'ın ön-eğitim dil dağılımında yoktur. Yine de "English decoder kalitesi" ve seq2seq kapasitesi nedeniyle transfer denenebilir, ancak tokenizasyon ve veri biçimi nedeniyle pratikte ByT5 kadar "doğal uyumlu" olmayabilir. [51]

İleri seviye mimari/strateji

Çoklu checkpoint ve ensemble / model soup: Public LB'de 34–35 bandına çıkan yaklaşımlarda, birden fazla ByT5 checkpoint'ini birleştirme (ağırlık ortalaması veya ensemble) sık görülür. Ağırlık ortalaması, inference maliyetini "tek model"e yakın tutarken ensemble etkisi sağlayabilir. [52]

Chunked decoding ve uzunluk yönetimi: ByT5 byte seviyesinde çalıştığı için sekanslar uzayabilir; bu nedenle max_length sınırları ve "uzun inputları parçalama" stratejileri pratikte önem kazanır. [53]

Named-entity koruma / kısıtlı decoding fikri: Resmî kanallarda adlandırılmış varlıkların (isimlerin) "öngörülemez" davranışı birincil engel olarak geçer ve onomasticon gibi yardımcı kaynakların kullanılabileceği belirtilir. Bu yarışmada, decoding sırasında "isimleri olduğu gibi kopyalamaya" eğilimli bir post-process katmanı (veya veri augmentasyonu) metrik açısından yüksek kaldıraçlı olabilir. [54]

Transfer learning mi, saf fine-tune mu?

Bu problem tipik olarak **transfer learning** problemidir: 1.561 örnekle sıfırdan büyük bir Transformer eğitmek rasyonel değildir; pre-trained bir seq2seq modelin fine-tune edilmesi beklenir. Transformer temeli ve seq2seq çerçevesi "Attention Is All You Need" çizgisinden gelir; T5/ByT5 gibi modeller de bu paradigmanın metin→metin görevlerinde transfer gücünü gösterir. [55]

Loss fonksiyonu ve yarışmaya özgü tweak'ler

Standart yaklaşım, **teacher-forcing altında çapraz entropi (token-level cross-entropy)** loss'udur. Bu yarışmada özel olarak önerilen iki tweak:

- **Label smoothing** (hem genel Transformer literatüründe hem de pratikte MT'de genelleme için faydalı olabilir). [56]
- **Format token'ları için bilinçli eğitim**: `<gap>`, `<big_gap>` gibi token'ların hem source'ta hem target'ta doğru "öğrenilmesi" için veri normalizasyonu; çünkü metrikte bu token'ların kaçırılması skor düşürür. [57]

Doğrulama, hiperparametre araması, inference ve ensembling

Sağlam validasyon kurgusu

Public LB'nin testin %34'ü olması, **LB overfit** riskini yükseltir; bu nedenle seçimleri CV'ye dayandırmak gerekir. [58]

Önerilen CV:

- **GroupKFold**: `oare_id` tablet/doküman kimliği gibi davranıyorsa, aynı `oare_id`'yi farklı fold'lara bölmemek için group split kullanın. Train küçük olduğu için 5-fold genellikle makuldür. [59]
- Eğer `oare_id` tekil ise, alternatif olarak stratification/genre gibi alanlar (varsa) üzerinden split düşünülebilir; ancak eldeki kanıtlar `oare_id` temelli ayrımın daha güvenli olduğunu işaret ediyor. [60]

Metrik hesaplaması için CV'de **corpus-level** BLEU ve chrF++ hesaplanmalıdır; tek tek cümle skorlarının ortalaması resmî sistemle denk olmayabilir. [61]

Hiperparametre araması planı

Public çözümlerde Optuna gibi araçlarla tuning örnekleri görülüyor; ancak internet kapalı olduğundan Optuna'nın kurulu olup olmadığı kontrol edilmelidir. [62]

Pratik bir arama uzayı:

- Learning rate: $[5e{-}5, 1e{-}4, 2e{-}4, 5e{-}4]$ (ByT5-base için daha düşük, small için biraz daha yüksek denenebilir) [63]
- Effective batch size: 16–128 (gradient accumulation ile) [64]
- Warmup ratio: 0.03–0.10 (lineer warmup + linear decay) [21]
- Epoch: 3–20 (küçük veri overfit riski nedeniyle early stopping önerilir) [65]
- Generation (eval için): `num_beams` 1/4/8; `length_penalty` 0.8–1.2; `max_new_tokens` 64–256 [66]

Inference, ensembling ve post-processing

Beam search: yüksek skor notebook'larında beam search yaygın. [67]

Checkpoint averaging (model soup): Aynı mimaride (ör. ByT5) birkaç fine-tune checkpoint'ini ağırlıklı ortalama ile birleştirip tek model gibi generate etmek, Kaggle'da süre/memory açısından iyi bir kompromidir. [45]

Gap token garanti post-process: Source'ta `<gap>` / `<big_gap>` varsa ve model kaçırıyorsa, skora zarar vermemek için çıktı sonuna eksik token'ları eklemek (opsiyonel) bir "metrik savunması" olabilir. Bu tür metrik-uyumlu hack'lerin konuşulduğu görülüyor; ancak private genelleme riski vardır. [68]

Teslimatlar ve çalıştırılabilir kod

Aşağıdaki dosyalar, Kaggle Notebook içinde `/kaggle/working/` altına yazılıp **doğrudan çalıştırılabilir** şekilde tasarlanmıştır. İnternet kapalı olduğu için `model_name_or_path` parametresini bir **yerel path** olarak vermeniz gerekir (ör. yarışma veri paketinin içindeki model klasörü veya ekleyeceğiniz bir Kaggle Dataset). Yarışma veri paketinde `/models/` altında ByT5 checkpoint'lerine referans veren notebook'lar bulunduğu görülüyor. [69]

Model karşılaştırma tablosu

Model ailesi	Tokenizasyon seviyesi	Bu yarışmada beklenen artı	Beklenen eksi / risk	Ne zaman seçilmeli
ByT5 (small/ base/large)	Byte-level (tokenizer-free) [70]	Gürültü/diakritik/gap gibi bozulmalara dayanıklılık; toplulukta yüksek public skorlar [66]	Byte dizileri uzun → hız/memory maliyeti; generation yavaşlayabilir [71]	Baseline ve "top-6 hedefli" ana omurga
mBART	SentencePiece subword [72]	Düşük kaynaklı MT'de transfer kazançları raporlanmış [73]	Kaynak dil dağılımı uyumsuz; transliteration formatına hassas olabilir	Deneysel ikinci model / ensemble çeşitliliği
MarianMT	Subword (model kartına bağlı) [74]	Hazır MT mimarisi; hızlı inference starter	Kaynak dil "modern dil" değil; performans plafonu düşük kalabilir	Hızlı sanity-check baseline
M2M / diğer multilingual MT	Subword	Modern MT altyapısı; karşılaştırma	Public skor örneklerinde ByT5 gerisinde kalabiliyor	Alternatif/ ensemble çeşitliliği

utils.py

```
# utils.py
# Kaggle offline uyumlu yardımcı fonksiyonlar (ön işleme, metrik, checkpoint
averaging)

from __future__ import annotations

import math
import os
import re
import json
import random
import unicodedata
from dataclasses import dataclass
from typing import Dict, Iterable, List, Optional, Sequence, Tuple

import numpy as np

# sacrebleu Kaggle'da çoğunlukla yüklü gelir; değilse önceden dataset olarak
ekleyin.
import sacrebleu


_GAP_RE = re.compile(r"\b[xX]{2,}\b")              # 'xx', 'xxx' gibi
token'lar
```

```
_BIG_GAP_RE = re.compile(r"\b[xX]{4,}\b")          # daha büyük boşluk
_ELLIPSIS_RE = re.compile(r"(\.\.\.|…{1,})")       # '...' veya '…'
_WS_RE = re.compile(r"\s+")


def seed_everything(seed: int = 42) -> None:
    random.seed(seed)
    np.random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)


def normalize_unicode(text: str) -> str:
    # NFKC bazı diakritikleri bozabilir; NFC daha güvenli.
    return unicodedata.normalize("NFC", text)


def normalize_whitespace(text: str) -> str:
    text = text.replace("\u00A0", " ")  # NBSP
    text = _WS_RE.sub(" ", text)
    return text.strip()


def normalize_transliteration(
    text: str,
    do_gap_normalization: bool = True,
    keep_case: bool = True,
) -> str:
    """
    Yarışma verisindeki transliteration'lar farklı geleneklerden gelebilir.
    Bu fonksiyon, mümkün olduğunca güvenli (semantiği bozmayan) bir
normalizasyon yapar.
    """
    if text is None:
        return ""

    text = normalize_unicode(text)

    if not keep_case:
        text = text.lower()

    # Bazı dosyalarda determinatifler farklı parantezlerle gelebiliyor.
    # Bu dönüşümleri agresif yapmıyoruz; yalnızca whitespace/Unicode + gap
standardizasyonu.
    if do_gap_normalization:
        # big gap önce
        text = _BIG_GAP_RE.sub("<big_gap>", text)
        text = _GAP_RE.sub("<gap>", text)

        # '...' ve unicode ellipsis
        # Hepsi <gap> demek her zaman doğru olmayabilir, ama genelde hasar/
boşluk gösterimi.
```

```
        text = _ELLIPSIS_RE.sub("<gap>", text)

        # Çoklu gap tokenlarını tekleştir
        text = re.sub(r"(?:<gap>\s*){2,}", "<gap> ", text)
        text = re.sub(r"(?:<big_gap>\s*){2,}", "<big_gap> ", text)

    text = normalize_whitespace(text)
    return text


def normalize_translation(text: str) -> str:
    if text is None:
        return ""
    text = normalize_unicode(text)
    text = normalize_whitespace(text)
    return text


def ensure_gap_tokens_present(source: str, pred: str) -> str:
    """
    Opsiyonel savunma: source'ta gap token'ı varsa ve pred'de yoksa ekle.
    Bu her durumda doğru değildir; fakat bazı çözümler metrik için bunu bir
"tamir" adımı olarak kullanır.
    """
    src_has_gap = "<gap>" in source
    src_has_big = "<big_gap>" in source
    if src_has_gap and "<gap>" not in pred:
        pred = pred.strip() + " <gap>"
    if src_has_big and "<big_gap>" not in pred:
        pred = pred.strip() + " <big_gap>"
    return normalize_whitespace(pred)


def compute_bleu_chrfpp(
    predictions: Sequence[str],
    references: Sequence[str],
) -> Dict[str, float]:
    """
    Yarışma metriğini yaklaşık olarak yeniden üretir:
      gm = sqrt( BLEU * chrF++ )

    Not:
    - sacrebleu BLEU/chrF skorlarını 0-100 aralığında döndürür, biz 0-1'e
ölçekliyoruz.
    - chrF++ için word_order=2 kullanıyoruz.
    """
    preds = [normalize_translation(p) for p in predictions]
    refs = [normalize_translation(r) for r in references]

    bleu = sacrebleu.corpus_bleu(preds, [refs]).score / 100.0
    chrfpp = sacrebleu.corpus_chrf(preds, [refs], word_order=2).score / 100.0
```

```
    gm = math.sqrt(max(bleu, 0.0) * max(chrfpp, 0.0))
    return {"bleu": bleu, "chrfpp": chrfpp, "gm": gm}


def average_checkpoints(
    checkpoint_dirs: Sequence[str],
    weights: Optional[Sequence[float]] = None,
) -> Dict[str, np.ndarray]:
    """
    Aynı mimariye ait birden çok HF checkpoint'inin pytorch_model.bin /
model.safetensors ağırlıklarını
    state_dict seviyesinde ortalar.

    Bu fonksiyon yalnız state_dict üretir; safe formatlar için
inference.py'de yükleme stratejisi uygulanır.
    Basitlik için burada torch import etmeden numpy tabanlı tutuyoruz;
inference tarafında torch ile yükleme yapılır.

    Not: Büyük modellerde memory maliyeti yüksek olabilir.
    """
    if weights is None:
        weights = [1.0 / len(checkpoint_dirs)] * len(checkpoint_dirs)
    if len(weights) != len(checkpoint_dirs):
        raise ValueError("weights ve checkpoint_dirs uzunluğu aynı olmalı.")

    # torch'a bağlı kalmadan: inference.py içinde torch.load ile state_dict
alınacak.
    return {"checkpoint_dirs": list(checkpoint_dirs), "weights":
list(weights)}  # placeholder spec


def save_json(path: str, obj: dict) -> None:
    with open(path, "w", encoding="utf-8") as f:
        json.dump(obj, f, ensure_ascii=False, indent=2)
```

train.py

```
# train.py
# ByT5 odaklı eğitim scripti (baseline + tuned modlar için parametrik).
# Kaggle Notebook içinde: !python train.py --help

from __future__ import annotations

import argparse
import os
from typing import Dict, List

import pandas as pd
import torch
```

```
from datasets import Dataset
from transformers import (
    AutoModelForSeq2SeqLM,
    AutoTokenizer,
    DataCollatorForSeq2Seq,
    Seq2SeqTrainer,
    Seq2SeqTrainingArguments,
)

from utils import (
    compute_bleu_chrfpp,
    normalize_transliteration,
    normalize_translation,
    seed_everything,
)


def parse_args() -> argparse.Namespace:
    p = argparse.ArgumentParser()

    p.add_argument("--data_dir", type=str, default="/kaggle/input/deep-past-
initiative-machine-translation")
    p.add_argument("--train_path", type=str, default=None)

    p.add_argument("--model_name_or_path", type=str, required=True)
    p.add_argument("--output_dir", type=str, default="/kaggle/working/
mt_model")

    p.add_argument("--seed", type=int, default=42)

    # veri kolonu isimleri (farklı notebook'larda farklı isimler olabiliyor)
    p.add_argument("--source_col", type=str, default="transliteration")
    p.add_argument("--target_col", type=str, default="translation")
    p.add_argument("--group_col", type=str, default="oare_id")

    # split
    p.add_argument("--val_ratio", type=float, default=0.1)

    # uzunluk
    p.add_argument("--max_source_length", type=int, default=512)
    p.add_argument("--max_target_length", type=int, default=256)

    # eğitim
    p.add_argument("--num_train_epochs", type=float, default=5.0)
    p.add_argument("--learning_rate", type=float, default=2e-4)
    p.add_argument("--weight_decay", type=float, default=0.01)
    p.add_argument("--warmup_ratio", type=float, default=0.06)

    p.add_argument("--per_device_train_batch_size", type=int, default=4)
    p.add_argument("--per_device_eval_batch_size", type=int, default=8)
```

```
    p.add_argument("--gradient_accumulation_steps", type=int, default=4)
    p.add_argument("--max_grad_norm", type=float, default=1.0)

    p.add_argument("--fp16", action="store_true")
    p.add_argument("--bf16", action="store_true")

    # generation (eval sırasında)
    p.add_argument("--num_beams", type=int, default=4)
    p.add_argument("--generation_max_new_tokens", type=int, default=128)

    # logging/saving
    p.add_argument("--logging_steps", type=int, default=50)
    p.add_argument("--save_total_limit", type=int, default=2)

    return p.parse_args()


def train_val_split(df: pd.DataFrame, val_ratio: float, seed: int) ->
(pd.DataFrame, pd.DataFrame):
    # GroupKFold eklemek isterse kullanıcı burada kolayca değiştirebilir.
    df = df.sample(frac=1.0, random_state=seed).reset_index(drop=True)
    n_val = max(1, int(len(df) * val_ratio))
    val_df = df.iloc[:n_val].reset_index(drop=True)
    trn_df = df.iloc[n_val:].reset_index(drop=True)
    return trn_df, val_df


def make_hf_dataset(df: pd.DataFrame, source_col: str, target_col: str) ->
Dataset:
    # HF Dataset
    return Dataset.from_pandas(df[[source_col, target_col]])


def main() -> None:
    args = parse_args()
    seed_everything(args.seed)

    train_path = args.train_path or os.path.join(args.data_dir, "train.csv")

    df = pd.read_csv(train_path)
    if args.source_col not in df.columns or args.target_col not in
df.columns:
        raise ValueError(f"Beklenen kolonlar yok. Kolonlar:
{df.columns.tolist()}")

    # minimal & güvenli normalizasyon
    df[args.source_col] = df[args.source_col].astype(str).apply(lambda x:
normalize_transliteration(x, do_gap_normalization=True))
    df[args.target_col] =
df[args.target_col].astype(str).apply(normalize_translation)
```

```
    trn_df, val_df = train_val_split(df, args.val_ratio, args.seed)

    tokenizer = AutoTokenizer.from_pretrained(args.model_name_or_path)

    model = AutoModelForSeq2SeqLM.from_pretrained(args.model_name_or_path)

    def preprocess_batch(batch: Dict[str, List[str]]) -> Dict[str,
List[int]]:
        inputs = batch[args.source_col]
        targets = batch[args.target_col]

        model_inputs = tokenizer(
            inputs,
            max_length=args.max_source_length,
            truncation=True,
        )

        with tokenizer.as_target_tokenizer():
            labels = tokenizer(
                targets,
                max_length=args.max_target_length,
                truncation=True,
            )

        model_inputs["labels"] = labels["input_ids"]
        return model_inputs

    trn_ds = make_hf_dataset(trn_df, args.source_col,
args.target_col).map(preprocess_batch, batched=True,
remove_columns=[args.source_col, args.target_col])
    val_ds = make_hf_dataset(val_df, args.source_col,
args.target_col).map(preprocess_batch, batched=True,
remove_columns=[args.source_col, args.target_col])

    data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model)

    def compute_metrics(eval_pred):
        preds, labels = eval_pred
        if isinstance(preds, tuple):
            preds = preds[0]

        decoded_preds = tokenizer.batch_decode(preds,
skip_special_tokens=True)

        labels = [[(t if t != -100 else tokenizer.pad_token_id) for t in
seq] for seq in labels]
        decoded_labels = tokenizer.batch_decode(labels,
skip_special_tokens=True)

        metrics = compute_bleu_chrfpp(decoded_preds, decoded_labels)
        return metrics
```

```
training_args = Seq2SeqTrainingArguments(
    output_dir=args.output_dir,
    do_train=True,
    do_eval=True,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    save_total_limit=args.save_total_limit,
    load_best_model_at_end=True,
    metric_for_best_model="gm",
    greater_is_better=True,

    learning_rate=args.learning_rate,
    weight_decay=args.weight_decay,
    warmup_ratio=args.warmup_ratio,

    num_train_epochs=args.num_train_epochs,
    per_device_train_batch_size=args.per_device_train_batch_size,
    per_device_eval_batch_size=args.per_device_eval_batch_size,
    gradient_accumulation_steps=args.gradient_accumulation_steps,

    max_grad_norm=args.max_grad_norm,

    fp16=args.fp16,
    bf16=args.bf16,

    predict_with_generate=True,
    generation_num_beams=args.num_beams,
    generation_max_new_tokens=args.generation_max_new_tokens,

    logging_steps=args.logging_steps,
    report_to="none",
)

trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=trn_ds,
    eval_dataset=val_ds,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

trainer.train()

# En iyi modeli yaz
trainer.save_model(args.output_dir)
tokenizer.save_pretrained(args.output_dir)

# Son değerlendirme
```

```
    final_metrics = trainer.evaluate()
    print("Final metrics:", final_metrics)


if __name__ == "__main__":
    main()
```

inference.py

```
# inference.py
# Kaggle code competition inference: test.csv -> submission.csv
# - test.csv örnek olabilir; skorlamada hidden test ile değişir.
# - Bu script "transliteration" kolonu varsa direkt kullanır.
# - Yoksa published_texts.csv üzerinden text_id + line_start/line_end ile
source üretmeyi dener (heuristic).

from __future__ import annotations

import argparse
import os
from typing import List, Optional, Tuple

import pandas as pd
import torch
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

from utils import (
    ensure_gap_tokens_present,
    normalize_transliteration,
    normalize_translation,
)


def parse_args() -> argparse.Namespace:
    p = argparse.ArgumentParser()

    p.add_argument("--data_dir", type=str, default="/kaggle/input/deep-past-
initiative-machine-translation")
    p.add_argument("--test_path", type=str, default=None)
    p.add_argument("--published_texts_path", type=str, default=None)

    p.add_argument("--model_name_or_path", type=str, required=True)

    p.add_argument("--batch_size", type=int, default=16)
    p.add_argument("--max_source_length", type=int, default=512)
    p.add_argument("--max_new_tokens", type=int, default=128)
    p.add_argument("--num_beams", type=int, default=4)

    p.add_argument("--gap_fix", action="store_true", help="Source'taki gap
token'larını çıktıda garanti etmeye çalışır (opsiyonel).")
```

```
    p.add_argument("--output_path", type=str, default="/kaggle/working/
submission.csv")

    return p.parse_args()


def _find_col(cols: List[str], candidates: List[str]) -> Optional[str]:
    for c in candidates:
        if c in cols:
            return c
    return None


def build_sources_from_test(
    test_df: pd.DataFrame,
    data_dir: str,
    published_texts_path: Optional[str],
) -> List[str]:
    cols = test_df.columns.tolist()

    # En basit senaryo: transliteration direkt testte var
    if "transliteration" in cols:
        return test_df["transliteration"].astype(str).tolist()

    # Aksi halde published_texts üzerinden üretmeyi dene
    pub_path = published_texts_path or os.path.join(data_dir,
"published_texts.csv")
    pub = pd.read_csv(pub_path)
    pub_cols = pub.columns.tolist()

    # join key tahmini
    test_key = _find_col(cols, ["text_id", "text_uuid", "oare_id"])
    pub_key = _find_col(pub_cols, ["text_id", "text_uuid", "oare_id"])

    if test_key is None or pub_key is None:
        raise ValueError(
            f"test.csv transliteration içermiyor ve join key bulunamadı.\n"
            f"test kolonları: {cols}\n"
            f"published_texts kolonları: {pub_cols}"
        )

    # transliteration kolonları
    pub_text_col = _find_col(pub_cols, ["transliteration",
"transliteration_orig"])
    if pub_text_col is None:
        raise ValueError(f"published_texts içinde transliteration kolonu
bulunamadı. Kolonlar: {pub_cols}")

    pub_map = dict(zip(pub[pub_key].astype(str).tolist(),
pub[pub_text_col].astype(str).tolist()))
```

```
    # slicing alanları
    line_start_col = _find_col(cols, ["line_start", "Line_start",
"linestart"])
    line_end_col = _find_col(cols, ["line_end", "Line_end", "lineend"])

    sources: List[str] = []
    for _, row in test_df.iterrows():
        k = str(row[test_key])
        full_text = pub_map.get(k, "")

        if line_start_col and line_end_col:
            ls = int(row[line_start_col])
            le = int(row[line_end_col])
            lines = full_text.split("\n")

            # Heuristic: önce [ls:le] dene; boşsa [ls:le+1]
            chunk = "\n".join(lines[ls:le])
            if len(chunk.strip()) == 0 and le + 1 <= len(lines):
                chunk = "\n".join(lines[ls:le + 1])
            if len(chunk.strip()) == 0:
                # fallback: tüm metin
                chunk = full_text
            sources.append(chunk)
        else:
            # slicing yoksa tüm metni kullan
            sources.append(full_text)

    return sources


@torch.inference_mode()
def generate_translations(
    model,
    tokenizer,
    sources: List[str],
    batch_size: int,
    max_source_length: int,
    max_new_tokens: int,
    num_beams: int,
) -> List[str]:
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model.eval()

    outputs: List[str] = []
    for i in range(0, len(sources), batch_size):
        batch_src = sources[i:i + batch_size]
        enc = tokenizer(
            batch_src,
            return_tensors="pt",
```

```
            padding=True,
            truncation=True,
            max_length=max_source_length,
        ).to(device)

        gen_ids = model.generate(
            **enc,
            num_beams=num_beams,
            max_new_tokens=max_new_tokens,
            early_stopping=True,
        )
        batch_out = tokenizer.batch_decode(gen_ids, skip_special_tokens=True)
        outputs.extend(batch_out)

    return outputs


def main() -> None:
    args = parse_args()

    test_path = args.test_path or os.path.join(args.data_dir, "test.csv")
    test_df = pd.read_csv(test_path)

    if "id" not in test_df.columns:
        raise ValueError(f"test.csv içinde 'id' kolonu yok. Kolonlar:
{test_df.columns.tolist()}")

    sources = build_sources_from_test(
        test_df=test_df,
        data_dir=args.data_dir,
        published_texts_path=args.published_texts_path,
    )

    # normalize source
    sources_norm = [normalize_transliteration(s, do_gap_normalization=True)
for s in sources]

    tokenizer = AutoTokenizer.from_pretrained(args.model_name_or_path)
    model = AutoModelForSeq2SeqLM.from_pretrained(args.model_name_or_path)

    preds = generate_translations(
        model=model,
        tokenizer=tokenizer,
        sources=sources_norm,
        batch_size=args.batch_size,
        max_source_length=args.max_source_length,
        max_new_tokens=args.max_new_tokens,
        num_beams=args.num_beams,
    )

    preds = [normalize_translation(p) for p in preds]
```

```
    if args.gap_fix:
        preds = [ensure_gap_tokens_present(src, p) for src, p in
zip(sources_norm, preds)]

    sub = pd.DataFrame({"id": test_df["id"], "translation": preds})
    sub.to_csv(args.output_path, index=False)
    print(f"Saved submission to: {args.output_path}")


if __name__ == "__main__":
    main()
```

README.md

```
# Deep Past Challenge – Reproducible MT Pipeline (ByT5-first)

Bu repo, Kaggle "Deep Past Challenge - Translate Akkadian to English"
yarışması için
(1) eğitim (train.py) ve (2) inference + submission üretimi (inference.py)
sağlar.

> Önemli: Bu bir Kaggle Code Competition. İnternet kapalı.
> Model dosyaları ve ek paketler (gerekirse) bir Kaggle Dataset olarak
notebook'a eklenmelidir.

## Veri yolları

Varsayılan:
- /kaggle/input/deep-past-initiative-machine-translation/train.csv
- /kaggle/input/deep-past-initiative-machine-translation/test.csv  (örnek
test; skorlamada hidden test ile değişir)
- /kaggle/input/deep-past-initiative-machine-translation/published_texts.csv
(testte transliteration yoksa slicing için)

## Çözüm 1: Hızlı ve güçlü baseline (tek model)

### Eğitim (isteğe bağlı)
Bu, küçük/orta bir ByT5 checkpoint'inden fine-tune yapar.

Notebook hücresi:
```bash
!python train.py \
  --model_name_or_path /kaggle/input/deep-past-initiative-machine-
translation/models/byt5-base-transformers-default-v1 \
  --output_dir /kaggle/working/byt5_baseline \
  --num_train_epochs 3 \
  --learning_rate 2e-4 \
  --per_device_train_batch_size 4 \
```
```

```
  --gradient_accumulation_steps 4 \
  --fp16
```

Inference + submission.csv

```
!python inference.py \
  --model_name_or_path /kaggle/working/byt5_baseline \
  --num_beams 4 \
  --max_new_tokens 128 \
  --batch_size 16 \
  --output_path /kaggle/working/submission.csv
```

Kaggle submission için dosya adı: `submission.csv`

Çözüm 2: Top-6 hedefli performans pipeline (CV + model soup + aggressive decoding)

Bu çözüm, pratikte şunları hedefler: - 5-fold eğitim (aynı script, farklı seed/split) - Her fold'dan en iyi checkpoint - Checkpoint averaging (model soup) veya multi-model inference - Beam search + (gerekirse) gap_fix

Not: Bu repo, "model soup" averaging için minimal altyapıyı bırakır; Kaggle ortamında 3-5 checkpoint'i ortalamak memory açısından pahalı olabilir. Önce 3 checkpoint ile başlayın.

Örnek plan: 1) 3 farklı seed ile aynı ayarlar: - seed=42, 777, 2026 2) Her birini /kaggle/working/ckpt_seedX olarak kaydet 3) En iyi 3'ünü seç ve ağırlık ortalaması / ensemble uygula 4) inference.py ile submission üret

Örnek inference (gap_fix açık):

```
!python inference.py \
  --model_name_or_path /kaggle/input/your_dataset_path/your_model_soup \
  --num_beams 8 \
  --max_new_tokens 160 \
  --batch_size 8 \
  --gap_fix \
  --output_path /kaggle/working/submission.csv
```

Metrik

CV'de: Score = sqrt(BLEU * chrF++) BLEU ve chrF++ corpus-level hesaplanır.

train.py içinde compute_metrics bunu otomatik raporlar.

Pratik notlar

- Public leaderboard testin sadece bir kısmı olabilir; CV'ye güvenin.
- Gap tokenları (<gap>, <big_gap>) stratejik olarak kritik olabilir.
- published_texts.csv üzerinden test slicing: test.csv transliteration içermiyorsa inference.py bunu dener. Eğer hidden test şeması farklıysa inference.py içinde join/slice heuristiklerini güncellemeniz gerekir. ```

Bu kod ve rehber, yarışma koşullarının (internet kapalı, örnek testin hidden test ile değişmesi, submission formatı, metrik tanımı) resmî açıklamalarıyla uyumlu şekilde tasarlanmıştır. [75]

---

[1] [29] [30] [32] [75] https://www.kaggle.com/competitions/deep-past-initiative-machine-translation
https://www.kaggle.com/competitions/deep-past-initiative-machine-translation

[2] [61] Deep Past Challenge - Translate Akkadian to English
https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/overview/evaluation?utm_source=chatgpt.com

[3] https://aclanthology.org/P02-1040/
https://aclanthology.org/P02-1040/

[4] chrF: character n-gram F-score for automatic MT evaluation
https://aclanthology.org/W15-3049/?utm_source=chatgpt.com

[5] [18] [35] Deep Past Challenge - Translate Akkadian to English
https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/672511?utm_source=chatgpt.com

[6] [9] [10] [12] [13] [24] [33] [40] https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/data
https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/data

[7] [28] [43] [48] [49] [53] [62] [66] [67] https://www.kaggle.com/code/harukiharada/byt5-optuna-tuning-chunked-beam-search
https://www.kaggle.com/code/harukiharada/byt5-optuna-tuning-chunked-beam-search

[8] [59] [60] https://www.kaggle.com/code/leiwong/deep-past-challenge-comprehensive-eda
https://www.kaggle.com/code/leiwong/deep-past-challenge-comprehensive-eda

[11] [42] https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/672511
https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/672511

[14] [19] [23] https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/664177
https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/664177

[15] https://www.kaggle.com/code/teruzuki061/akkadian-data-processing-updated-for-translations/log
https://www.kaggle.com/code/teruzuki061/akkadian-data-processing-updated-for-translations/log

[16] [17] [41] [54] https://www.linkedin.com/posts/deep-past-initiative_deep-past-challenge-translate-akkadian-activity-7413962172214497280-hNXN
https://www.linkedin.com/posts/deep-past-initiative_deep-past-challenge-translate-akkadian-activity-7413962172214497280-hNXN

[20] https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/670489
https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/670489

21 A Call for Clarity in Reporting BLEU Scores
https://aclanthology.org/W18-6319/?utm_source=chatgpt.com

22 39 57 68 https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/664948
https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/664948

25 63 70 71 https://arxiv.org/abs/2105.13626
https://arxiv.org/abs/2105.13626

26 https://aclanthology.org/D18-2012.pdf
https://aclanthology.org/D18-2012.pdf

27 https://aclanthology.org/P16-1162/
https://aclanthology.org/P16-1162/

31 64 https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/
https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/

34 58 65 https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/670467
https://www.kaggle.com/competitions/deep-past-initiative-machine-translation/discussion/670467

36 51 72 73 https://arxiv.org/abs/2001.08210
https://arxiv.org/abs/2001.08210

37 38 https://www.einpresswire.com/article/875885564/deep-past-initiative-partners-with-kaggle-on-prize-challenge-to-translate-4-000-year-old-tablets
https://www.einpresswire.com/article/875885564/deep-past-initiative-partners-with-kaggle-on-prize-challenge-to-translate-4-000-year-old-tablets

44 https://www.kaggle.com/code/sword4949/deep-past-challenge-ver-3
https://www.kaggle.com/code/sword4949/deep-past-challenge-ver-3

45 52 https://www.kaggle.com/code/anthonytherrien/byt-ensemble-script
https://www.kaggle.com/code/anthonytherrien/byt-ensemble-script

46 https://www.kaggle.com/code/artembredikhin/akkadian-m2m-inference
https://www.kaggle.com/code/artembredikhin/akkadian-m2m-inference

47 https://www.kaggle.com/code/meenalsinha/hybrid-best-akkadian
https://www.kaggle.com/code/meenalsinha/hybrid-best-akkadian

50 74 https://huggingface.co/docs/transformers/en/model_doc/marian
https://huggingface.co/docs/transformers/en/model_doc/marian

55 56 https://arxiv.org/abs/1706.03762
https://arxiv.org/abs/1706.03762

69 https://www.kaggle.com/code/carp0308/akkadian-mt-preprocessing-ensemble-inference
https://www.kaggle.com/code/carp0308/akkadian-mt-preprocessing-ensemble-inference