Review of Soham Kotkar’s Task Submission

What’s done well:

- Tokenizer Creation: SentencePiece tokenizer handles Hindi, Sanskrit, Marathi, and English; Unicode normalization for Devanagari applied.
- Model Integration: Light decoder-only model (GPT-NeoX/BLOOM-560M) successfully swapped with custom multilingual tokenizer.
- Language Routing: Basic language detection + routing in place (fastText/langdetect).
- API Endpoints: /tokenize, /generate, /language-detect implemented and tested on sample Gurukul queries.
- Stateless & Modular Design: Good for KB integration and future scaling.

Gaps / Areas for Improvement:

- Corpus Limitation: Currently limited to 4 languages; AI4Bharat corpora not loaded → need more robust Indic datasets for real-world usability.
- Evaluation Metrics Missing: No BLEU/ROUGE/MOS-proxy or human evaluation reported for grammar or tokenization quality.
- RLHF Prep: Ready, but no real reward model connection yet.
- Integration Hooks: Endpoint not yet integrated with full KB / Vaani / TTV pipeline.

Score: 8/10 – solid technical base, missing scale & evaluation for production-level deployment.

Next Step / Task Assignment

Task Name: Indic Multilingual Expansion & MCP Training

Objective: Extend Soham’s tokenizer + LM to cover at least 20 Indian languages, make the LM robust for real datasets, and prepare it for large-scale Gurukul integration.

Task Breakdown:

1. Language Expansion
    - Add 16+ additional Indian languages (Tamil, Telugu, Kannada, Bengali, Gujarati, Punjabi, Odia, Malayalam, Assamese, Marathi dialects, etc.).
    - Ensure proper Unicode normalization & sentencepiece merges.

    - Collect clean corpora: Wikipedia dumps, Indic NLP corpora, open datasets from AI4Bharat, HindMono, CC-100, OSCAR, or other public sources.

2. MCP (Multi-Corpus Preprocessing) Training
    - Preprocess corpora for consistent tokenization across scripts.
    - Train multilingual tokenizer on combined dataset (SentencePiece).
    - Save vocab & merges in sharable format.
3. LM Fine-tuning on Real Datasets
    - Fine-tune decoder-only LM (BLOOM/NeoX) using MCP datasets.
    - Validate grammar + sentence fluency for each language.
    - Ensure language switching mid-conversation works reliably.
4. Integration Prep
    - Ensure API endpoints accept 20+ languages.
    - Prepare wrapper for /generate and /tokenize to feed Vaani TTS.
    - Make output fully compatible with Indigenous NLP composer (Nisarg) and Vaani TTS (Karthikeya).
5. Evaluation & QA
    - Automatic evaluation: BLEU/ROUGE, perplexity, tokenization accuracy.
    - Manual checks for fluency in 5–10 prompts per language.
    - Latency checks to ensure API scales for multiple requests concurrently.

Deliverables:

- Multilingual tokenizer for ≥20 Indian languages.
- Fine-tuned LM compatible with new tokenizer.
- REST API for tokenization, generation, and language detection.
- Integration guide for linking with Indigenous NLP + Vaani TTS.
- Evaluation report (automatic + manual).

Recommendations for Robust System Design

1. Scalability:
    - Dockerize tokenizer + LM for cloud deployment; scale horizontally using FastAPI + GPU allocation.
    - Integrate caching for repeated tokenization/generation to reduce latency.
2. Extensibility:
    - Modular architecture → new languages added easily without retraining full LM.
    - Separate preprocessing, tokenization, and inference modules.
3. Data Pipeline Robustness:
    - Ensure cleaning of scripts, remove noise, handle transliterations, diacritics, and ligatures.
    - Automate corpora ingestion from open datasets + periodic updates.
4. User Experience:
    - Integrate LM output seamlessly with Vaani TTS → voice-native experience.
    - Ensure multi-turn context memory + language switching mid-dialogue.
5. Evaluation Metrics:
    - Use BLEU/ROUGE/Perplexity for text quality.
    - Track inference latency, memory usage, error logs.
6. Integration Path:
    - Provide structured API for Indigenous NLP composer → TTS → Gurukul UI.
    - Allow other teams (Nisarg, Karthikeya, Shashank) to plug in audio/video outputs easily.

Perfect. Here's a concise, structured, high-value learning kit for Soham to master MCP (Multi-Corpus Preprocessing) and efficiently implement it for our 20+ Indian language LM expansion:

MCP Learning Material for Soham

1. Conceptual Overview

Goal: Combine multiple corpora (parallel, monolingual, multilingual) into a unified, clean, tokenizable dataset for LM training while preserving grammar, semantics, and script integrity.

Core Concepts:

- Unicode Normalization: Ensure consistency across Devanagari, Tamil, Telugu, Bengali, etc.
- Script-specific token handling: Ligatures, diacritics, compound characters.
- Sentence Segmentation: Use language-specific rules to avoid splitting wrong boundaries.
- Deduplication & Noise Removal: Remove duplicates, irrelevant tokens, HTML tags, encoding errors.
- Tokenization Prep: SentencePiece or BPE-friendly formatting.

Reference Reads:

- Google's SentencePiece Paper: https://arxiv.org/abs/1804.10959
- AI4Bharat: Indic NLP preprocessing best practices: https://github.com/AI4Bharat/indicnlp

2. Step-by-Step MCP Workflow

1. Corpus Collection
    - Wikipedia dumps (Indic languages)
    - OSCAR/CC-100 multilingual corpora
    - AI4Bharat Indic corpora
    - Gurukul-specific curated text (lessons, scripts)
2. Cleaning & Normalization
    - Remove control characters, HTML tags, boilerplate text
    - Unicode normalization (NFC/NFKC)
    - Strip punctuation/noise for tokenization
    - Optional transliteration consistenc
3. Sentence Segmentation
    - Use indic-nlp-library for Indian languages
    - Tools: nltk, sacremoses, or custom regex-based split
4. Deduplication
    - Hash each sentence → remove duplicates

- Optional fuzzy matching for near-duplicates

5. Tokenization Preparation
    - Normalize whitespace & punctuation
    - Add language tags for multilingual corpora: <lang:hi> ... </lang>
    - Format ready for SentencePiece/BPE training
6. Training Tokenizer
    - Use sentencepiece.SentencePieceTrainer.train()
    - Merge vocab across languages to handle multilingual LM
7. Integration
    - Save vocab + merges → plug into LM
    - Test tokenization → detokenization loop for all language

3. Recommended Tools & Libraries

Tool	Purpose
sentencepiece	Train multilingual tokenizer
indic-nlp-library	Tokenization, normalization, transliteration
fasttext	Language detection for routing
regex	Script-specific cleaning
transformers (Hugging Face)	LM integration, decoding
pandas/numpy	Corpus processing, deduplication
aiofiles / asyncio	Fast I/O for large corpora

4. Sample MCP Pipeline Snippet (Python)

```
import sentencepiece as spm
from indicnlp.normalize.indic_normalize import
IndicNormalizerFactory
import re

# 1. Normalize Indic text
factory = IndicNormalizerFactory()
normalizer = factory.get_normalizer("hi")  # for Hindi
```

```
text = "नमस्ते, यह एक उदाहरण है।"
normalized = normalizer.normalize(text)

# 2. Clean noise
clean_text = re.sub(r'[^0-9a-zA-Z\u0900-\u097F\s]', '',
normalized)

# 3. Save corpus for SentencePiece
with open("corpus.txt", "w", encoding="utf-8") as f:
    f.write(clean_text + "\n")

# 4. Train SentencePiece tokenizer
spm.SentencePieceTrainer.train(
    '--input=corpus.txt --model_prefix=indic_sp --
vocab_size=32000 --character_coverage=1.0 --model_type=bpe'
)
```

5. Learning Resources / References

1. Hugging Face: Training Tokenizers
    - https://huggingface.co/docs/tokenizers/python/latest/
2. AI4Bharat: Indic NLP Preprocessing
    - https://github.com/AI4Bharat/indicnlp
3. SentencePiece Guide
    - https://github.com/google/sentencepiece
4. Unicode & Devanagari Handling
    - https://unicode.org/
    - https://www.unicode.org/reports/tr29/
5. Python Text Processing Tutorials
    - re, unicodedata, ftfy for cleaning

6. Tips for Fast, Optimized MCP Usage

- Batch Processing: Read/write corpora in chunks to avoid memory issues.
- Parallelization: Use multiprocessing or joblib for preprocessing large corpora.

- Language Tags: Prepend language codes (<hi>, <ta>, <bn>) to help LM distinguish scripts.
- Cache Intermediate Steps: Save normalized & cleaned files for reproducibility.
- Validation: Random sample checks for tokenization quality per language.

Note to Soham

Hi Soham,

Good progress on the multilingual LM base — the tokenizer and initial LM integration for Hindi, Sanskrit, Marathi, and English is solid.

Next, we need to expand this to at least 20 Indian languages using MCP pipelines and real-world datasets, so our system can serve a truly pan-Indian audience. Focus on clean preprocessing, robust tokenization, and API readiness for Indigenous NLP + Vaani TTS integration.

Once completed, this will become the backbone of our multilingual capabilities, making Gurukul scalable for millions of users. Keep logging evaluation metrics and integration notes so handoff is smooth.