用户提到的两种推理模式，独立运行和推测解码，表面上看似矛盾，但实际上可能有不同的应用场景或配置。我需要理解这两种模式的具体实现及其背后的设计逻辑。

1. **独立运行模式**：

- 丢弃MTP模块，仅使用主模型生成。这意味着模型完全依赖主模型的参数和能力，按传统的自回归方式逐个生成令牌。

- 这种模式的优势在于简单、稳定，不需要额外的模块参与，适合对生成速度要求不高但需要高准确性的场景。

2. **推测解码模式**：

- 复用MTP模块来加速生成。推测解码通常指利用辅助模型（或模块）生成多个候选令牌，然后由主模型快速验证这些候选，以减少主模型的调用次数，从而提升吞吐量。

- 用户提到的85%-90%的接受率，说明MTP模块生成的候选令牌大部分被主模型接受，验证有效，因此可以显著减少生成时间。

用户疑惑的点在于，为何在一种模式下需要丢弃MTP模块，而在另一种模式下却要复用。可能的解释是，这两种模式针对不同的需求进行优化：

- **独立运行模式**适用于对延迟不敏感但需要确保最高生成质量的场景。此时，不使用MTP模块可以避免任何潜在的干扰或错误传播，确保每一步生成都经过主模型的完整计算。

- **推测解码模式**适用于对吞吐量要求较高的场景，通过利用MTP模块生成候选令牌，主模型只需验证这些候选，从而减少计算量，提升生成速度。尽管MTP模块生成的候选可能不完全准确，但高接受率（85%-90%）保证了加速效果的有效性。

接下来，我需要验证这种理解是否正确，并查看是否有文献或技术文档支持这种设计。推测解码（Speculative Decoding）在大型语言模型中的应用确实存在，例如使用较小的草案模型生成候选，主模型进行验证，从而加速生成。DeepSeek-V3可能将MTP模块作为草案模型的一部分，在需要加速时复用该模块，而在不需要加速时仅使用主模型。

此外，需要确认MTP模块在推理阶段的具体作用。MTP模块在训练时用于多令牌预测，但在推理时可能被重新利用为草案生成器，生成多个候选令牌供主模型验证。这种情况下，MTP模块的参数和[illegible]能在两种模式中的不同应用并不矛盾，而是根据需求灵活配置。