طرق تقليص Overhead أثناء التشغيل

1. Pre-JIT / Precompilation

- **Ngen** : يحوّل IL إلى native code قبل التشغيل، يقلل تأخير JIT في بداية التشغيل
- **ReadyToRun (R2R** : البادیل في+5 .NET Core/.NET. ، يقلل تأخير JIT لكنه أقل كفاءة قليلاً من NGen

2. Tiered Compilation & AOT Partial

- **Tiered JIT** :يبدأ بتجميع سريع ثم يعيد تحسين الأكواد الحرارية (hot) +5/NET Core.بيستخدم
- و **CoreRT** يدعم AOT كامل، لكن مش مفعّل إفتراضياً

3. Profile-Guided Optimization (PGO)

- يجمع معلومات تشغيل فعلية، يُحسّن قرارات JIT (inlining,hoisting...)
- في +6 NET. ممكن استخدام **Dynamic**و **Static PGO** لتحسين الأداء عبر التجارب.

4. Multicore JIT

- باستخدام System.Runtime.ProfileOptimization، CLRيسجل الأكواد المطلوبة ويشغّل JIT بالتوازي على عدة نوى، مما يقلل وقت الإقلاع (16–35% تحسن)

5. PrepareMethod و PrepareMethod

- باستخدام [Prep] من System.Runtime.CompilerServices لإعداد ميثود معينة مسبقاً لمنع JIT وقت.

6. تحسينات JIT الداخلية

- **RyuJIT**شهد تطور مستمر devirtualization :، loop optimizations، ...hoistingتحسينات متقدمة في.NET Framework & Core.
- في **NET 10.**تم إجراء تحسينات عميقة: تقليل استدعاء الواجهات، تخصيص الذاكرة في الستاك، والـ loop hoisting.

التقنية	الهدف	التأثير
NGen / R2R	للتقليل من تأخير البداية pre-JIT	يقلل وقت الإقلاع
Tiered JIT	تجميع مرحلي لتسريع الكود الحراري	أداء متوازن وسريع
PGO	تحسين بواسطة بيانات تشغيل حقيقية	واستخدام الذاكرة liningتحسين في
Multicore JIT	تجميع متوازي لتسريع الإقلاع	يقلل بنسبة 16–35٪ من وقت startup
PrepareMethod	pre-load لـ hot methods	عند استخدامها JIT runtime تقليل
RyuJIT & .NET 10	تحسينات تحت الغطاء للكود المرشح	أداء أفضل تلقائيًا

الخلاصة:

لتقليل تأخير JIT وقت التشغيل:

- استخدم **Pre-JIT**عبر NGen أو R2R.
- فعّل **Tiered & Multicore JIT**.
- جرّب **PGO**لتحسين الأداء عبر بيانات التشغيل.
- دع **RyuJIT**الجيل الحديث يمثل أداءً قويًا تلقائياً.