**محمدحسین صباغی ۹۹۲۳۰۴۴**

**گزارش پروژه ۳ هوش مصنوعی**

۱)

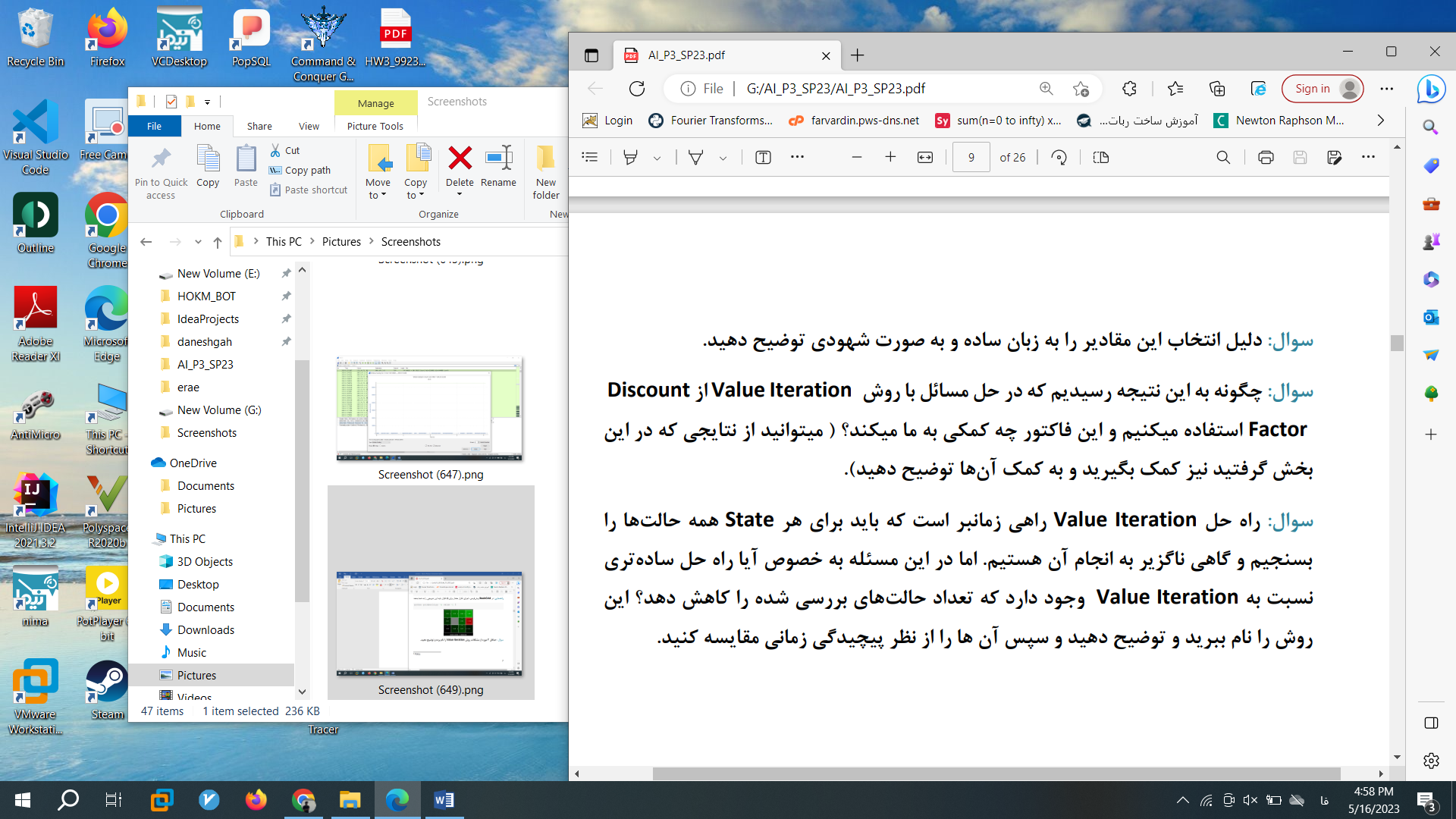


همگرا شدنش بسیار کند است و policy ها زودتر همگرا میشوند.

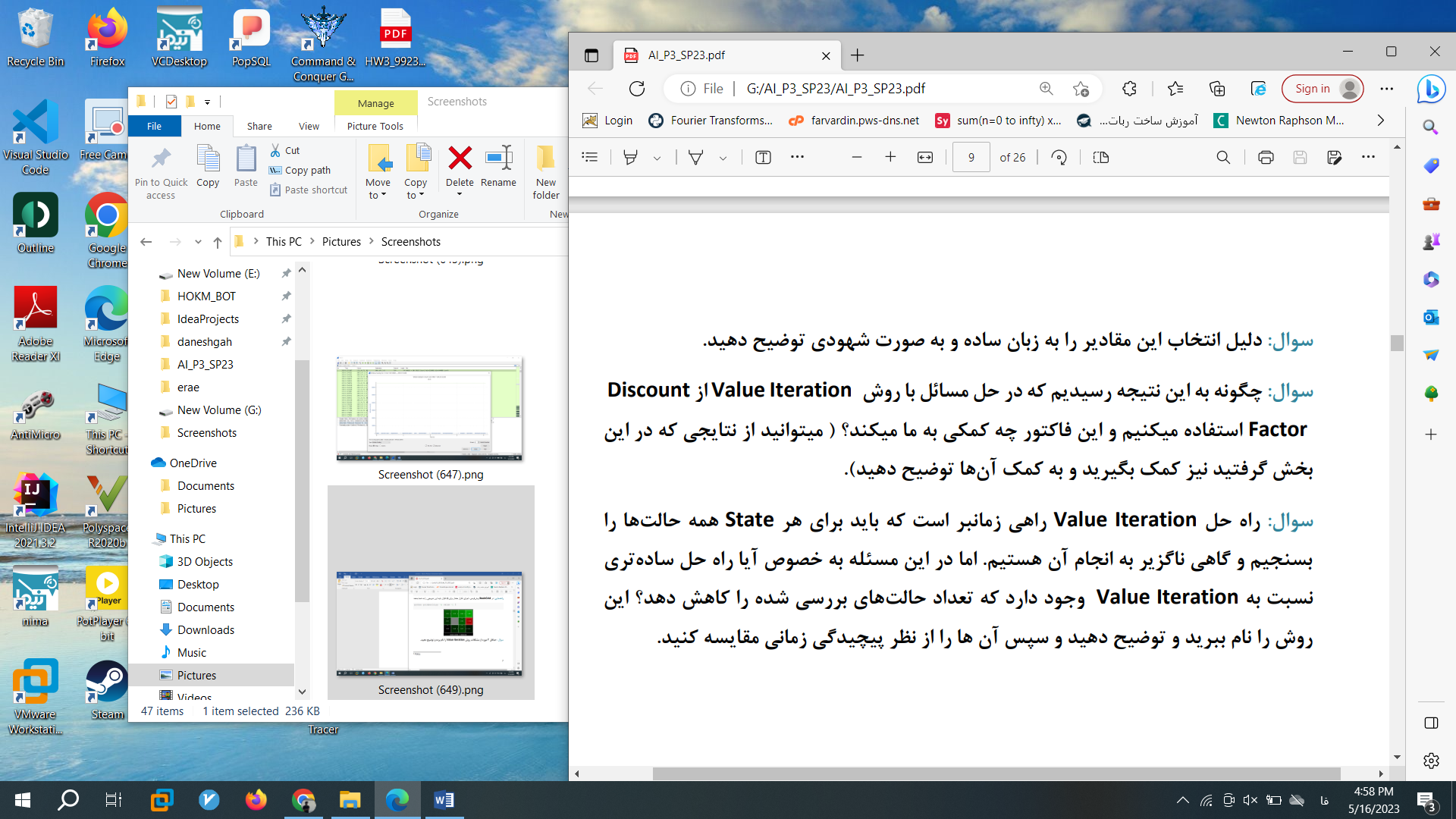
محاسبات زیادی دارد زیرا در هر مرحله تمام استیت ها را چک میکنیم.

ممکن است به اطلاعاتی از جمله استیت ها و احتمالات و ریوارد ها دسترسی نداشته باشیم.

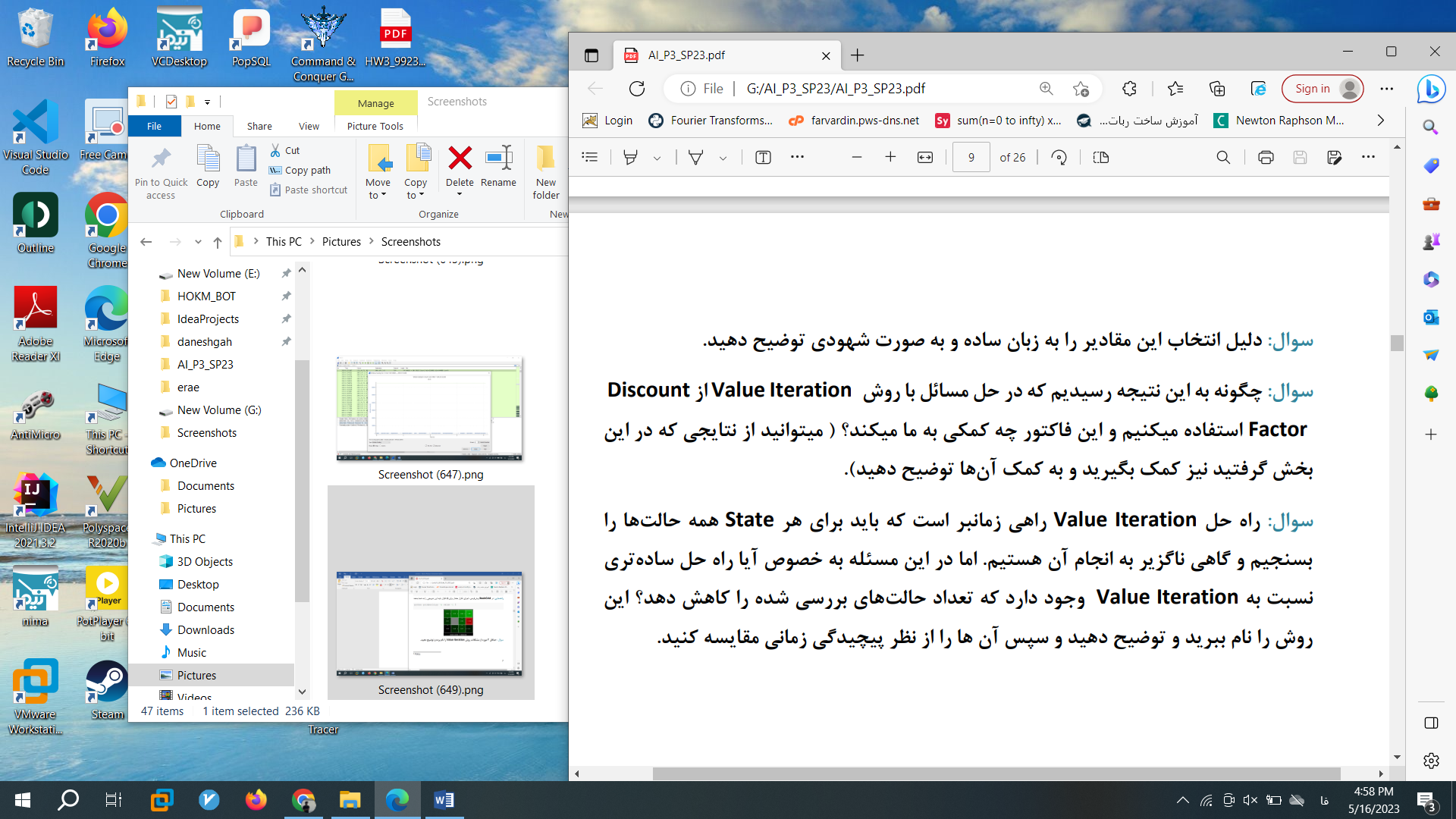
۲)



زیرا با کم شدن نویز تا میل کردن به صفر(یا حتی خود صفر) باعث میشود که با انتخاب هر جهتی ایجنت دقیقا به یک استیت خاص برود و به همین دلیل مقدار q انتخاب هایی که به سمت امتیاز ۱۰ میروند زیاد میشود و باعث میشود ایجنت سیاستش را به سمت آن رفتن بگذارد.



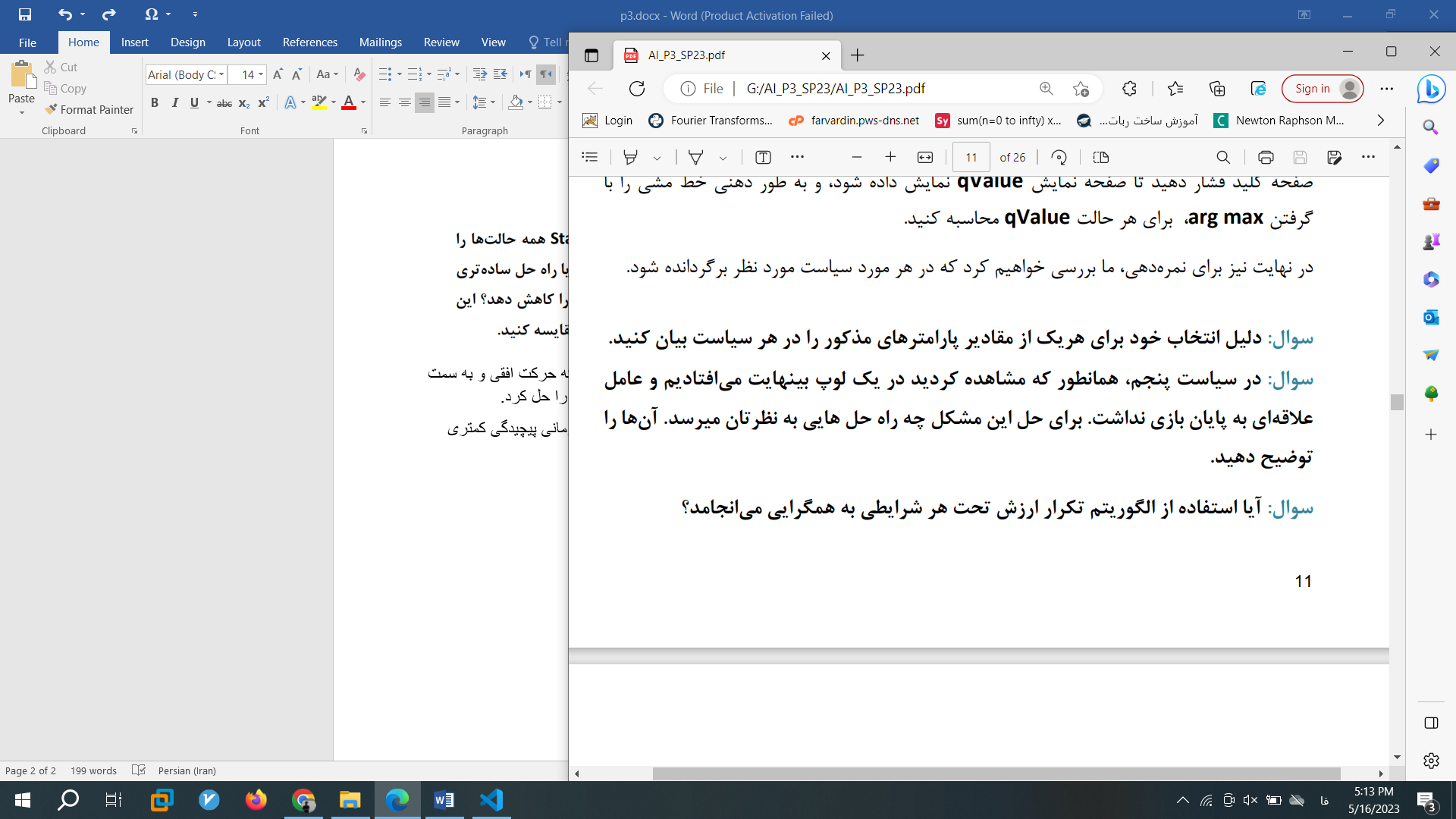
زیرا به دلیل وجود نویز بهتر است در حرکات کمتری امتیاز را دریافت کنیم زیرا با حرکات بیشتر احتمال خطا رفتن ایجنت به دلیل نویز بیشتر میشود. بنابراین با گذاشتن تخفیف ایجنت را تشویق میکنیم که زودتر امتیاز را دریافت کند.



بله در این مسیله استفاده از policy iteration بهتر است زیرا مشخص است که حرکت افقی و به سمت شرق بهتر است و با قرار دادن آن برای اکشن های اولیه میتوان سریعتر مسیله را حل کرد.

همانطور که مشخص است همگرایی این روش سریعتر است بنابراین از نظر زمانی پیچیدگی کمتری دارد.

۳)



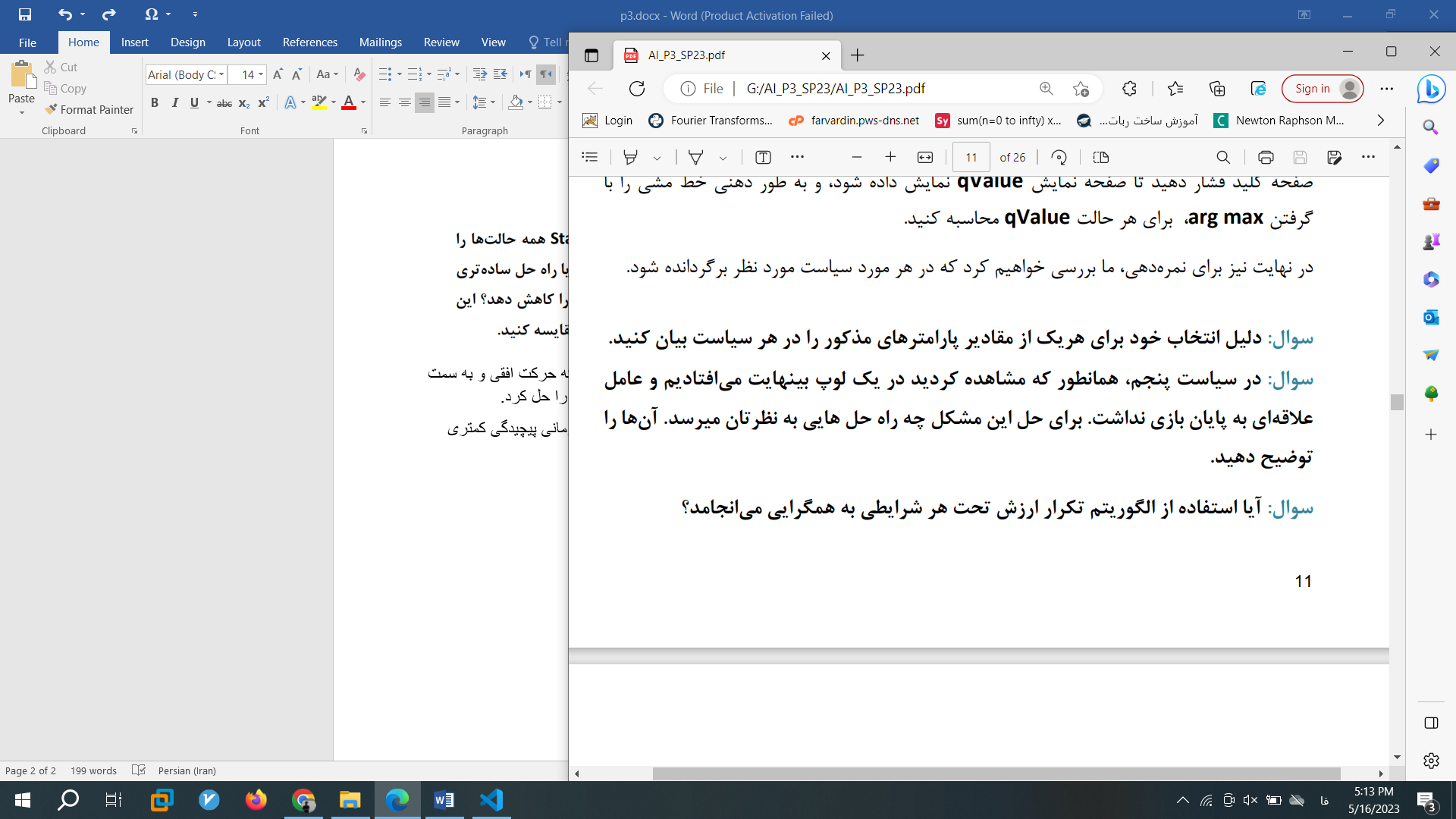
برای حالت اول مشخص است که با کم یا حتی صفر کردن نویز از سمت صخره ها میرود زیرا بعید است اشتباه برود و همچنین با زیاد کردن امتیاز منفی زنده ماندن و کاهش تخفیف باعث میشود که به سراغ ۱ که نزدیکتر است برود.

برای حالت دوم با زیاد کردن نویز احتمال خطا رفتن از سمت صخره ها زیاد میشود و ترجیح میدهد که از بالا برود و از صخره ها دوری کند.

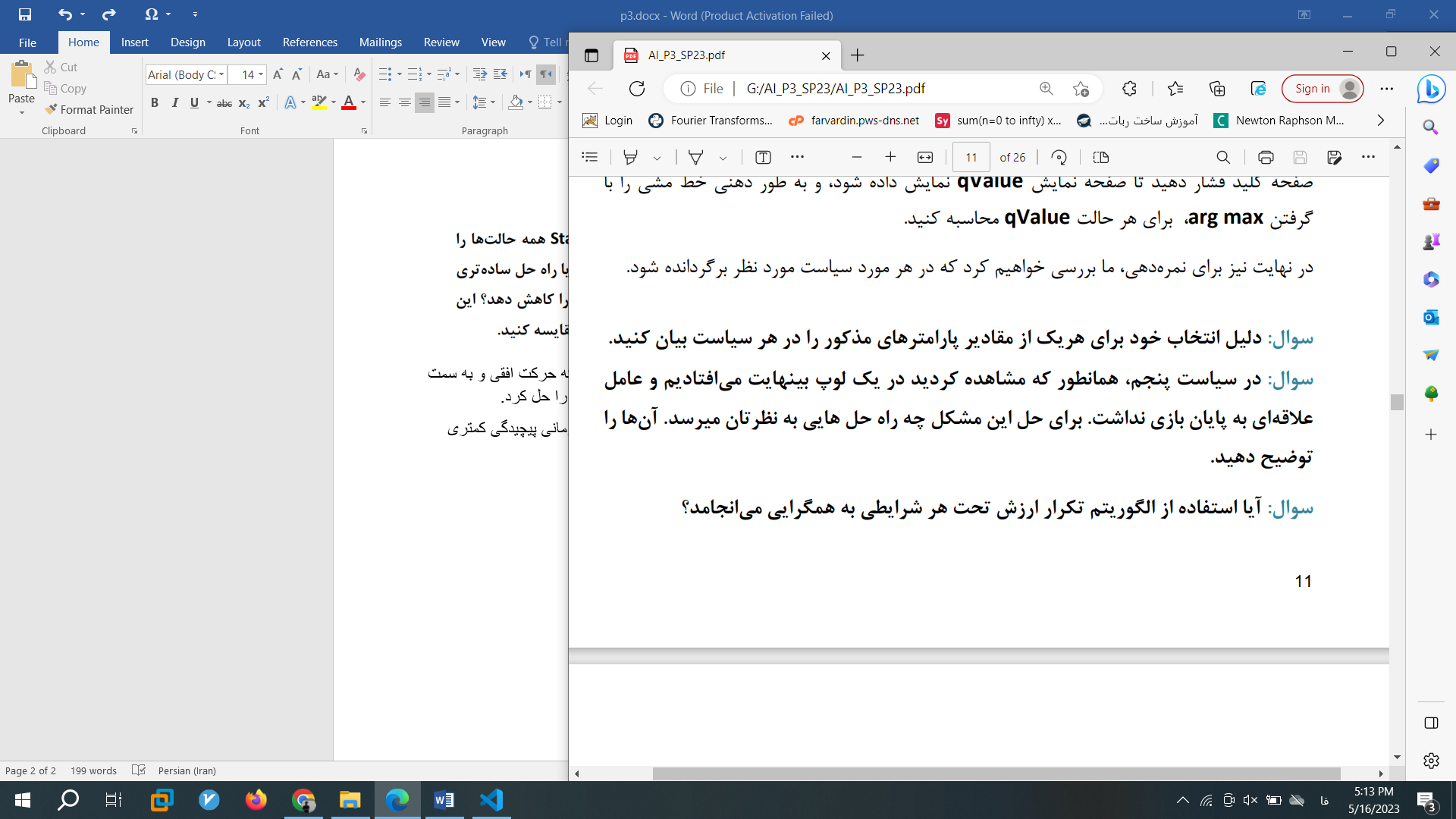
در حالت سوم با کاهش نویز و افزایش تخفیف باعث میشود که از پایین برود و به سمت امتیاز بیشتر برود زیرا تاثیر تخفیف کم است و در نتیجه امتیاز ۱۰ خیلی کم نمیشود.

برای حالت چهارم نویز را بیشتر میکنیم تا به خاطر احتمال خوردن به صخره از پایین نرود.

برای حالت پنجم با مثبت کردن امتیاز زنده موندن و صفر کردن نویز باعث میشود که تا بینهایت زنده بماند و امتیاز دریافت کند.

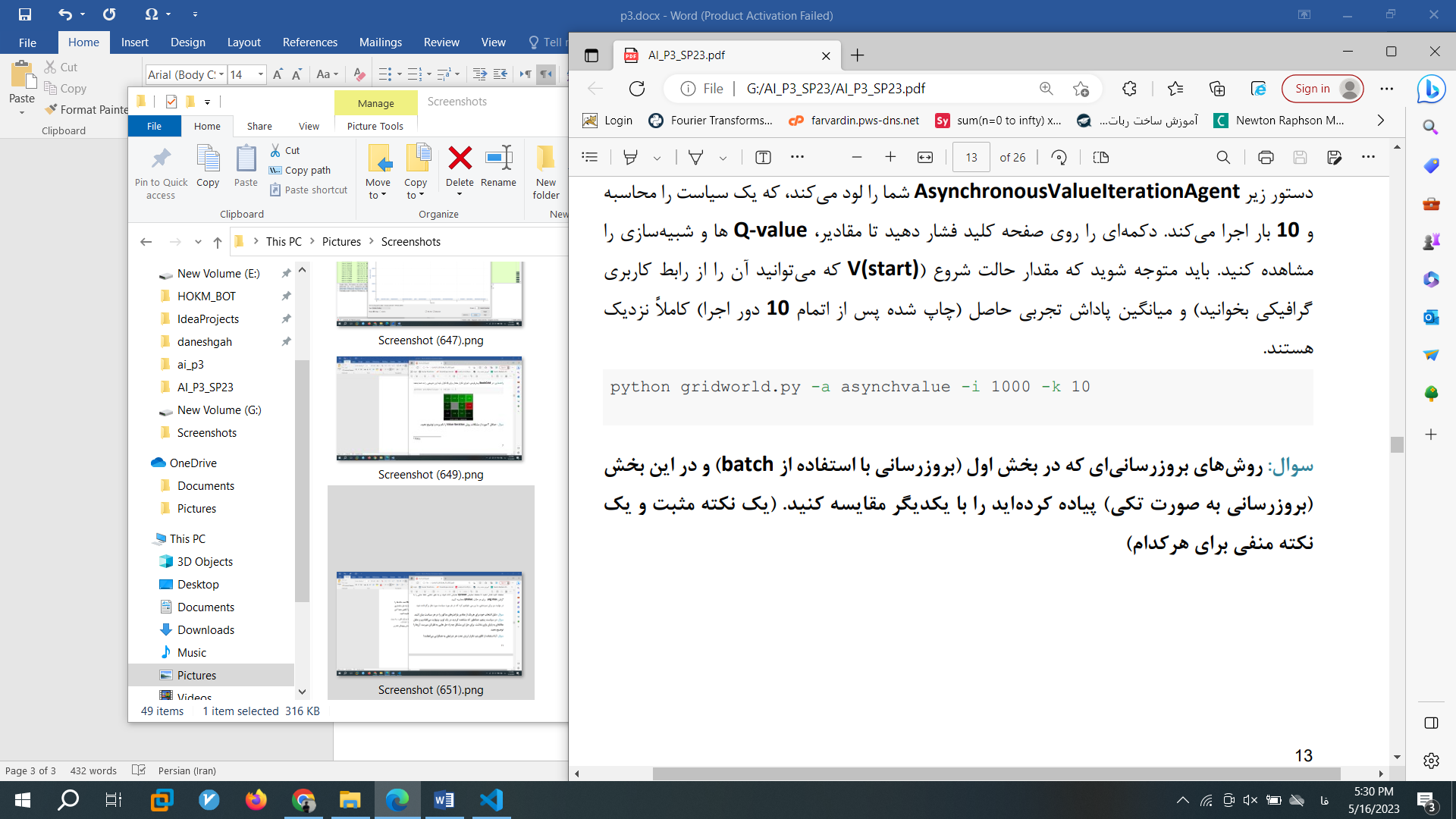


باید امتیاز زنده ماندن را منفی کنیم. یا متوانیم تا یک عمق محدودی را تحلیل کنیم که با اینکار ایجنت فقط تا یک تعداد حرکت خاصی را پیش بینی میکند و ترجیح میدهد امتیاز بزرگ پایان بازی را دریافت کند تا اینکه امتیاز های کم زنده ماندن تا حرکات محدود را دریافت کند.



اگر مقدار تخفیف برابر یک نباشد قطعا همگرا میشود زیرا از یک تعداد حرکتی به بعد اینقدر مقدار تخفیف کم میشود که ریوارد به صفر میل میکند و همگرا میشود.

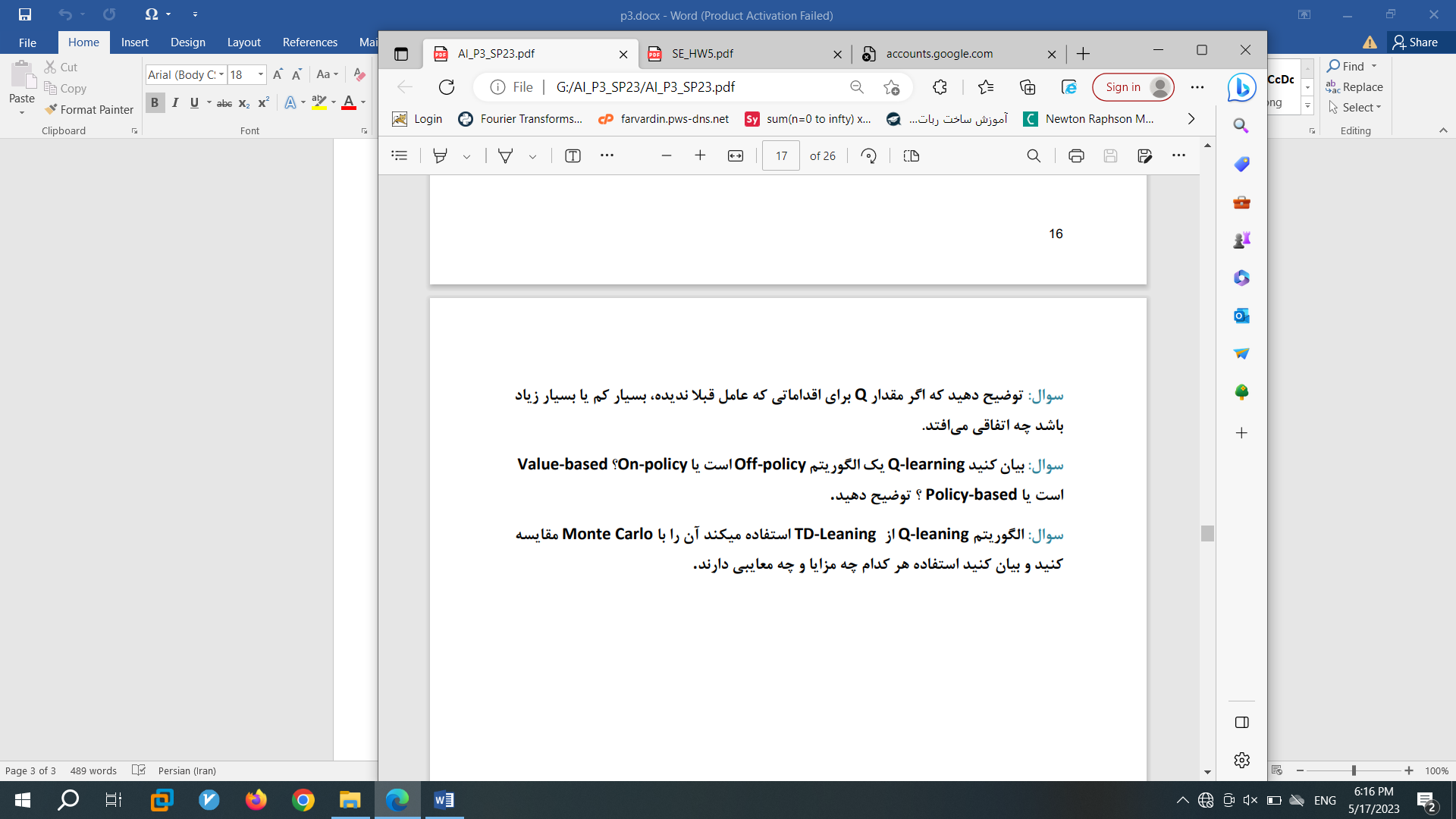
۴)



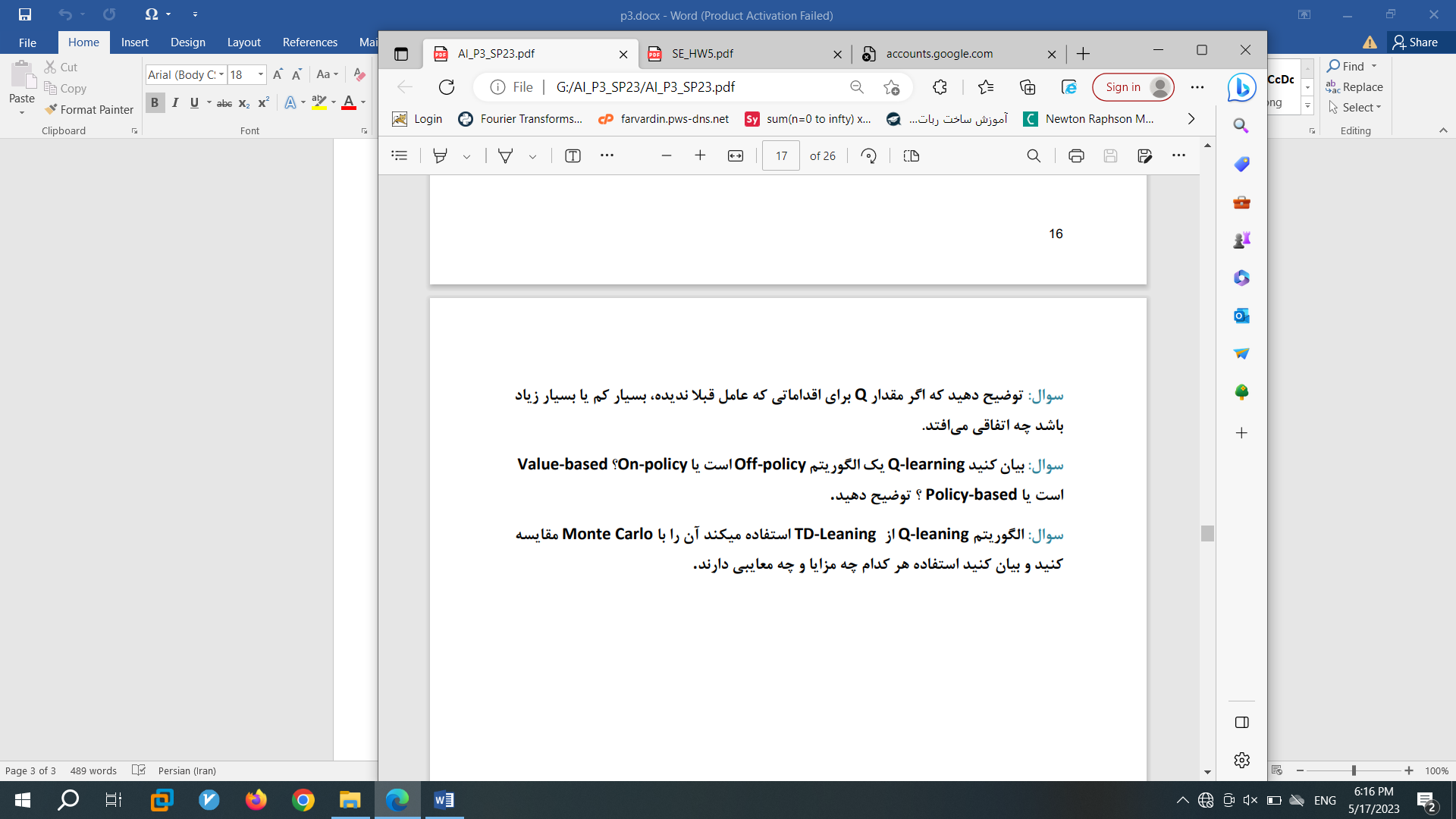
مزیت batch و عیب راه دوم : راه اول پیچیدگی کمتری دارد و تعداد تحلیل تمام استیت ها با هم برابر است.

عیب اولی و مزیت دومی: در حالت دوم امتیازها سریعتر همگرا میشوند زیرا با آپدیت شدن هر استیت سریعتر تاثیرش را روی بقیه استیت ها میگذارد و نیاز نیست تا یک دور کامل صبر کند.

۶)



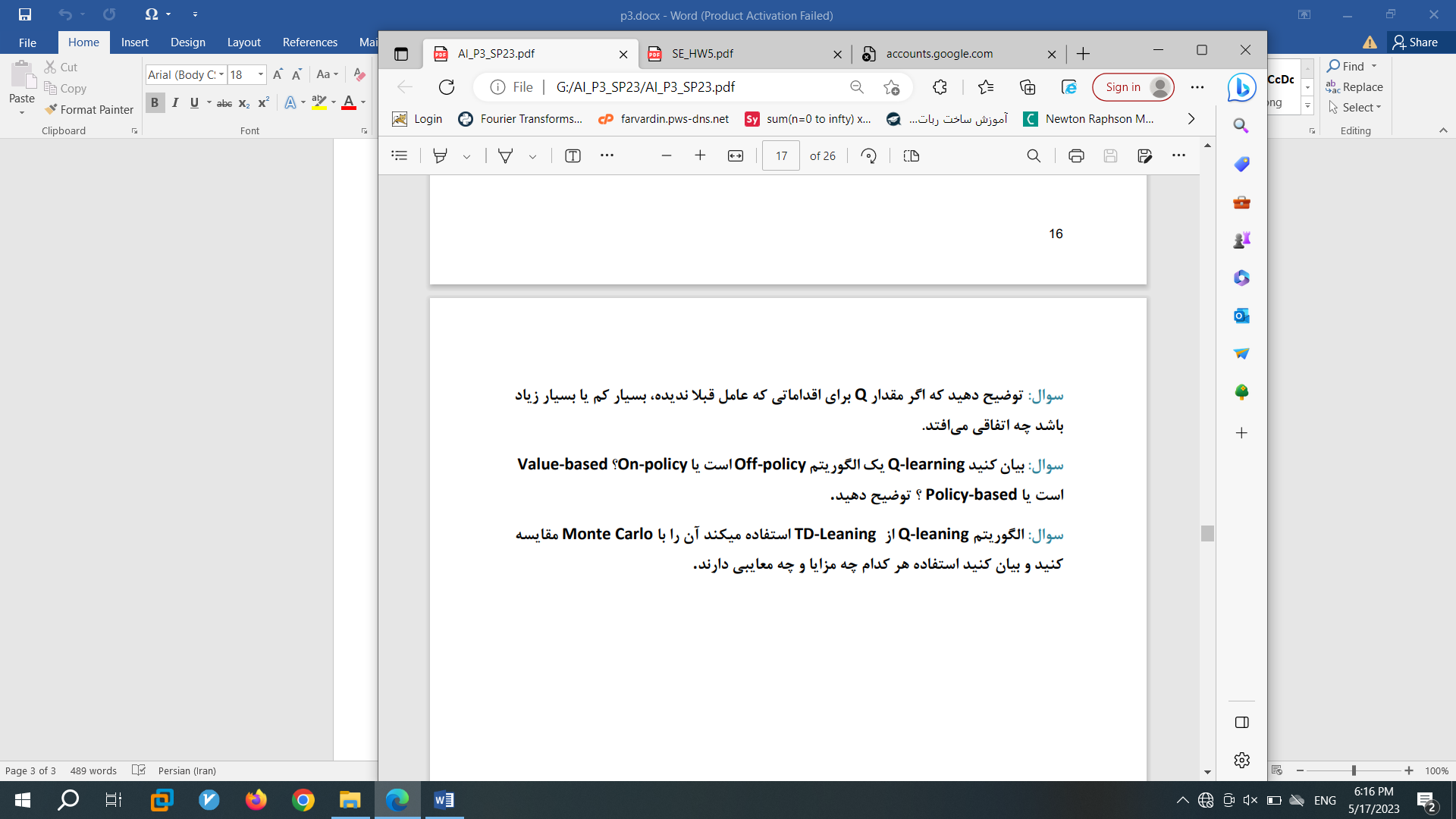
میتواند باعث شود در اپیزود های بعدی از رفتن به آن استیت ها خودداری کند یا اینکهبیشتر به سمت آن استیت ها برود و در هر دو مورد باعث میشود استیت های دیگری کمتر ویزیت شوند یا اصلا ویزیت نشوند.



Off-policy است زیرا policy که آپدیت میشود با policy رفتار متفاوت است.یعنی به دنبال بهینه تر کردن یک policy نیستیم بلکه به دنبال به دست آوردن یک policyهستیم.

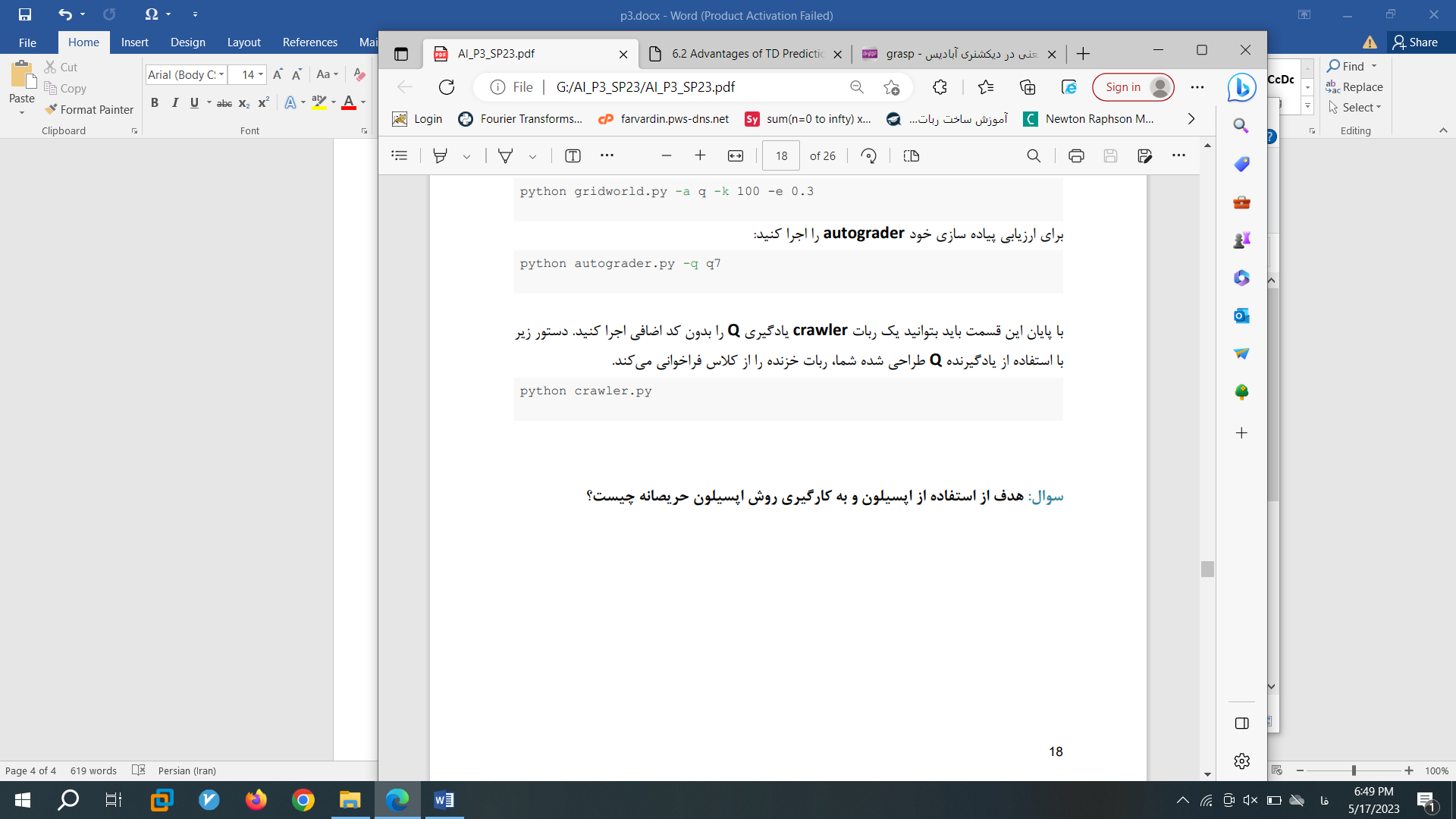
Value-based است زیرا به دنبال به دست آوردن q value ها و استفاده از آنها برای تصمیم گیری میباشد.

در td یا temporal difference بعد از هر گام ولیو ها را آپدیت میکنیم ولی در monte carlo پس از اتمام هر اپیزود آپدیت میکنیم.



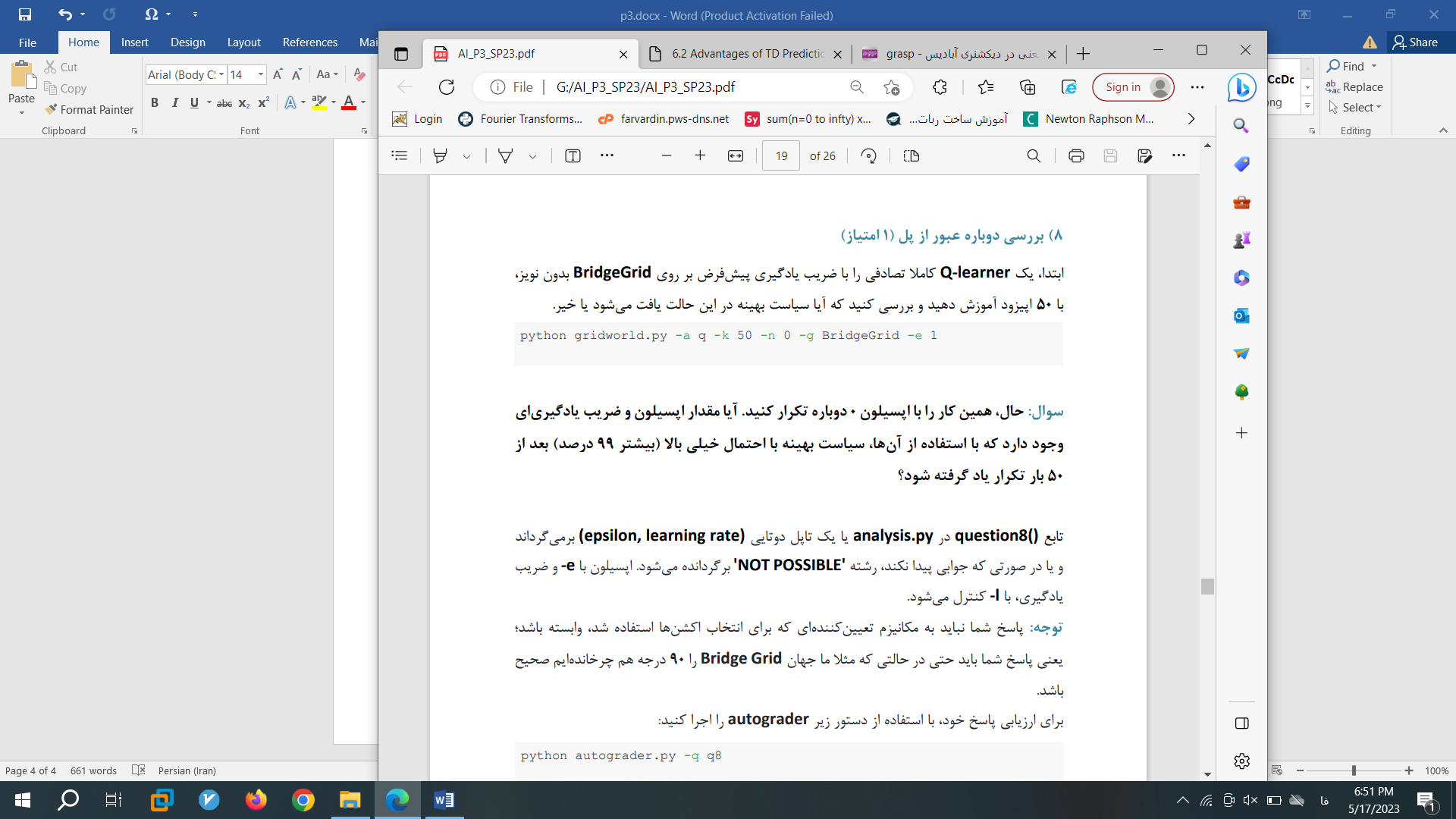
مزیت td اینست که سریعتر ولیوهای ما آپدیت میشوند و در نتیجه زودتر همگرا میشوند و همچنین از شرایط مارکوف طبعیت میکند یعنی امتیاز هر استیت صرفا به حال و اکشن آینده بستگی دارد ولی از طرفی هزینه بیشتری دارد و توان پردازشی بیشتری میخواهد ولی monte carlo ساده تر است ولی از مارکوف طبعیت نمیکند.

۷)

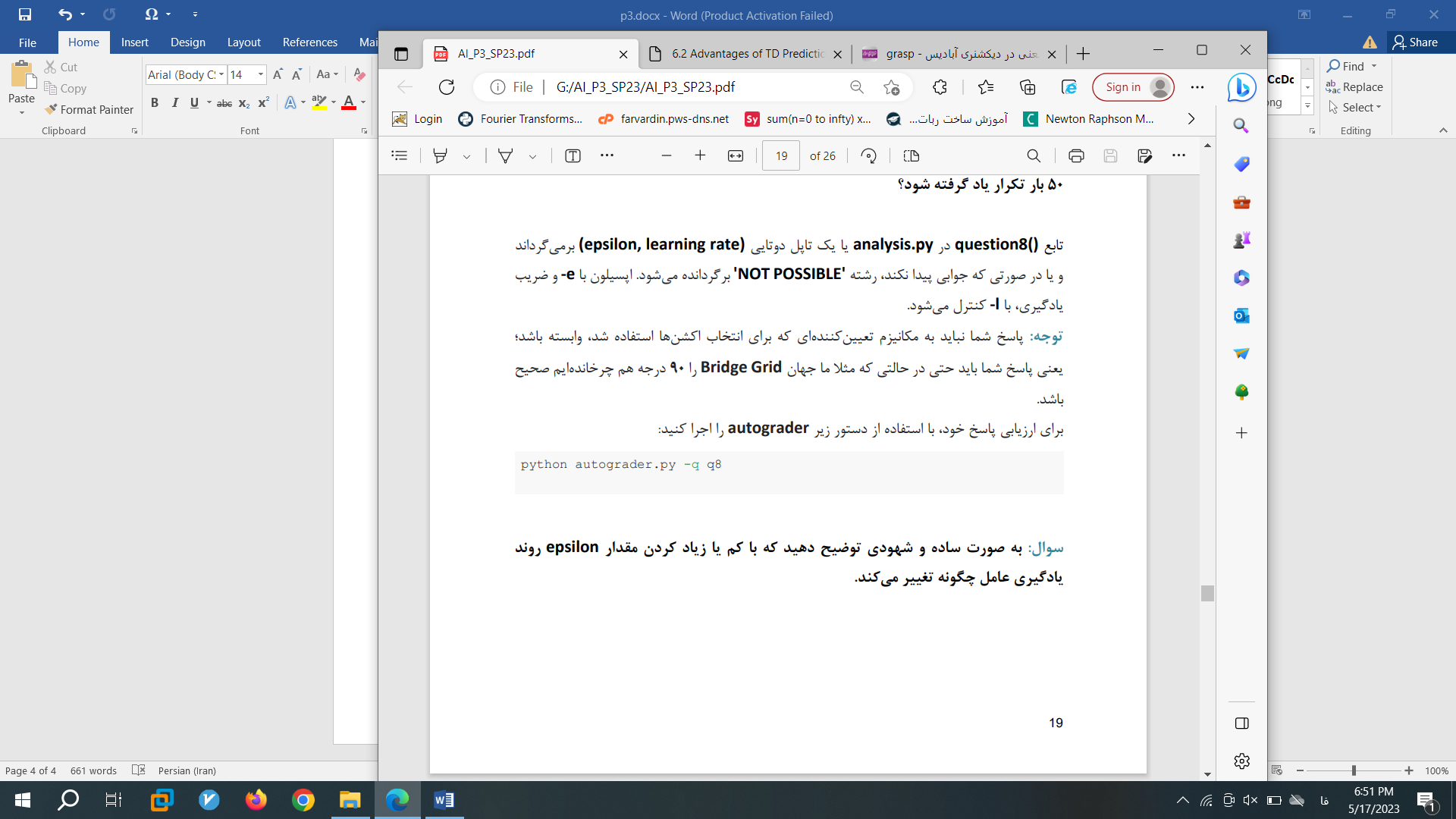


اینکار باعث میشود استیت های امتحان نشده را هم امتحان کند یا به عبارتی باعث exploration میشود و اگر از اپسیلون استفاده نکنیم ممکن است جواب بهینه به ما ندهد زیرا تمام استیت ها را امتحان نکرده است تا به بهینه برسد.

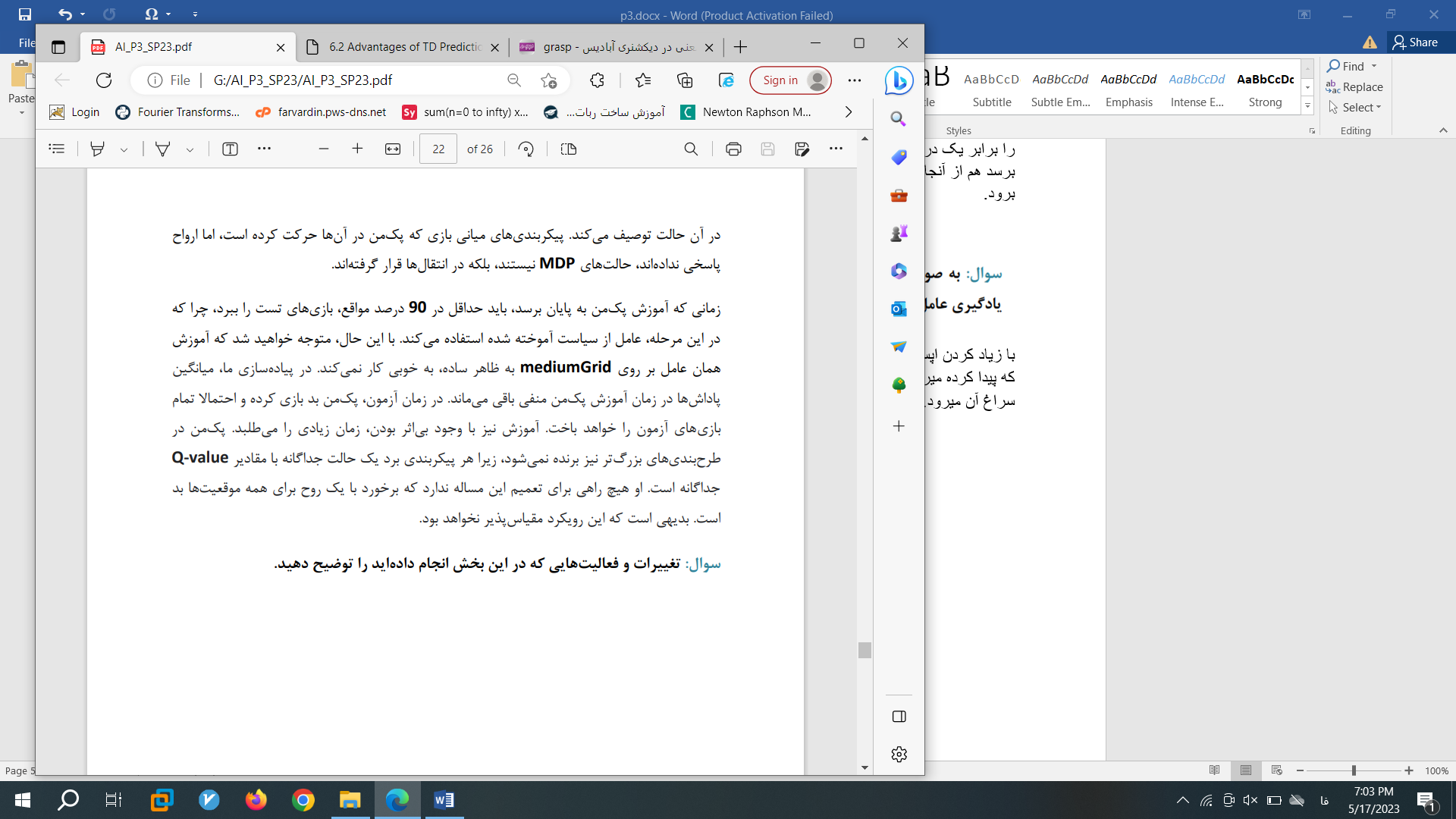
۸)



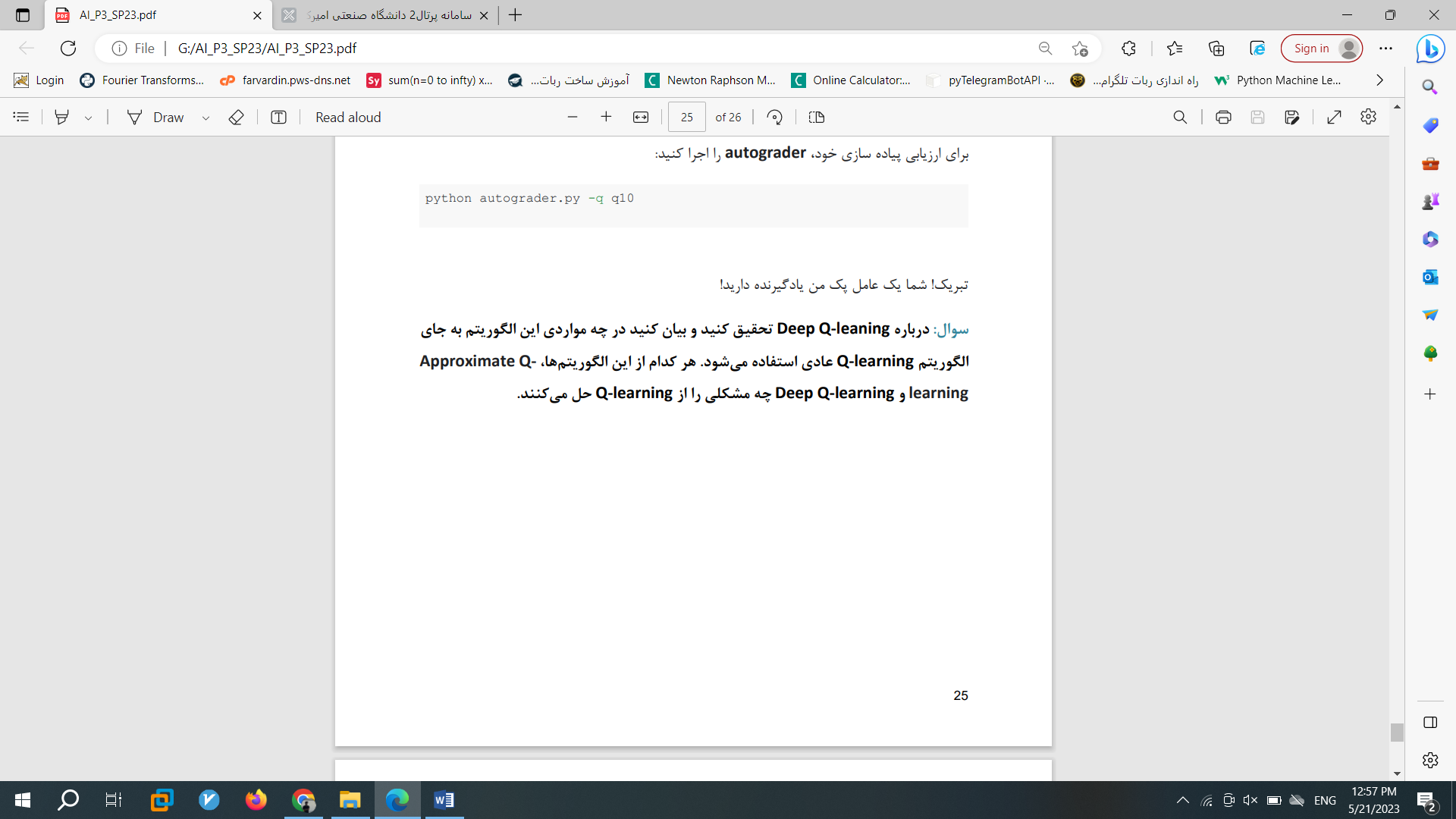
خیر زیرا باید چهار خانه متوالی به سمت راست حرکت کند تا به امتیاز بهینه برسد و حتی اگر اپسیلون را برابر یک درنظر بگیریم بازهم احتمال خیلی کمی دارد که به خانه سمت راست برسد و حتی اگر برسد هم از آنجایی که اپسیلون را صفر در نظر گرفتیم حتی پس از آن هم سعی نمیکند به سراغ آن برود.



با زیاد کردن اپسیلون استیت های جدید بیشتری را اکسپلور میکند ولی کمتر به سراغ policy بهینه ای که پیدا کرده میرود. با کم کردن اپسیلون کمتر اکسپلور میکند ولی اگر policy بهینه ای پیدا کند بیشتر به سراغ آن میرود. به طور کلی با افزایش اپیسلون مقدار اکسپلور بیشتر میشود.



هیچ تغییری نیاز نداشت و درست بود.



مانند q learning است با این تفاوت که با گرفتن یک استیت و با استفاده از شبکه های عصبی برای هر اکشن از آن استیت یک q value محاسبه میکند.

در مواردی که استیت ها خیلی زیاد است و پیچیدگی زیادی دارد نمیشود از q learning استفاده کرد و بهتر است از deep q learning استفاده کنیم.

Approximate q learning باعث کاهش سایز table در q learning نیشود.

Deep q learning پیچیدگی q learning برای مواردی که پیچیدگی خیلی زیادی دارند کاهش میدهد.